On the Construction and Upper Bounds of Balanced and Correlation-immune Functions

Markus Schneider
University of Hagen, Lehrgebiet Kommunikationssysteme,
58084 Hagen, Germany
email: mark.schneider@fernuni-hagen.de

Abstract: Correlation-immunity, as a property of Boolean functions, is an important criterion in the context of pseudo-random sequence generation for stream cipher systems. In this paper, an efficient algorithm for the construction of correlation-immune functions is given. It will be shown that the proposed algorithm provides a method to construct every mth order correlation-immune function. Besides correlation-immunity, also other properties of Boolean functions, like balance, can be taken into account in the construction. Because of the relevance in cryptographic applications, the focus will be on balanced correlation-immune functions. The complexity analysis of the proposed algorithm leads to a new upper bound for the number of balanced correlation-immune functions, depending on the number of input variables n and the order of correlation-immunity.

1 Introduction

Stream cipher systems use random sequences for the encryption of a message stream. In practical use, these random sequences are generated by a machine which works deterministically. These machines have to be well designed in order to avoid known methods of attacks, e.g. correlation attacks (see [Sieg 84], [Sieg 85]). Siegenthaler introduced the property of correlation-immunity for functions whose output do not give any information about their input to a certain degree. Correlation-immune functions are applied in the design of random sequence generators. The combiner generator is a well-analyzed machine for random sequence generation, because it allows the control of some cryptographically relevant properties like Linear Complexity [RuSt 85], for example. The combiner generator consists of n binary sources whose outputs are combined by a memoryless Boolean combiner function $f: GF(2)^n \rightarrow GF(2)$, $\underline{x} \rightarrow f(\underline{x})$. Each binary source is controlled by a secret key. If the combiner function is not properly chosen, there is a possibility to attack the key of each binary source separately and independently of the keys of the other binary sources. A memoryless function f is said to be correlation-immune of order m, with $1 \leq m \leq n$, if the output of f and any m input variables are statistically independent. More formally, if we think of the output of the binary source i, $1 \leq i \leq n$, to be modelled as a random variable X_i, a function f is mth order correlation-immune if the mutual information between the function output and any subset of m elements $X_{i_1}, \ldots, X_{i_m}$ of the n input variables is zero, with $1 \leq i_1 < \ldots < i_m$ [Sieg 84]:

$$I(f(X_1, \ldots, X_n); X_{i_1}, \ldots, X_{i_m}) = 0. \tag{1}$$

Ghuo-Zhen and Massey pointed out that the Walsh transform of an mth order correlation-immune function has some specific properties, which is sometimes used as alternative definition of mth order correlation-immune functions [GuMa 88]. Proposals for the construction of

correlation-immune functions were given in [Sieg 84], [CCCS 91] and [SeZh 93]. In this paper, a completely new approach to the construction of correlation-immune functions is presented.

Another relevant property of Boolean functions is balance. A function f is called balanced, if its output probabilities are equal, i.e. $P(f(\underline{x}) = 0) = P(f(\underline{x}) = 1) = 0.5$. Beside others, balance as property of the Boolean function is a necessary condition for the generation of a balanced sequence.

2 Relevant properties of correlation-immune functions

The following definition and theorem are of great importance in the context of the construction method of mth order correlation-immune functions presented in the next section. Only those properties of correlation-immune functions with paper related relevance are introduced.

Definition 2.1 *Let $n \geq 0$ be an integer. The set of all $(x_1, \ldots, x_n)$ with $x_i \in \{0,1\}$ is called an $n-$cube.*

Let I be a subset with k elements of the index set $\{1, \ldots, n\}$ of the $n-$cube: $I = \{i_1, \ldots, i_k\}$. For every $j \notin I$, let $a_j \in \{0,1\}$ be constant. The set $\{(x_1, \ldots, x_n)\}$ with $x_i \in \{0,1\}$ for $i \in I$ and $x_i = a_i$ fixed for $i \notin I$ will also be called a k-cube.

Theorem 2.1 *The function $f : GF(2)^n \rightarrow GF(2)$, $\underline{x} \rightarrow f(\underline{x})$ is mth order correlation-immune with $1 \leq m < n$, if and only if the binary output a, $a \in GF(2)$, appears in all $(n-m)$-dimensional subcubes of the $n-$cube with the same frequency K, $0 \leq K \leq 2^{n-m}$.*

Proof: $\Rightarrow$ Suppose a appears in all $(n-m)-$dimensional cubes with frequency K. Then, there is $P(f(\underline{x}) = a \mid x_{i_1} = a_1, \ldots, x_{i_m} = a_m) = K \cdot 2^{m-n}$, with $a_i \in GF(2)$. The frequency of a in the $n-$cube is obtained by summing up over all 2^m disjunct $(n-m)-$dimensional cubes. This yields $P(f(\underline{x}) = a) = 2^m \cdot K \cdot 2^{-n}$. Obviously, $P(f(\underline{x}) = a) = K \cdot 2^{m-n} = P(f(\underline{x}) = a \mid x_{i_1} = a_1, \ldots, x_{i_m} = a_m)$, which means, that the value of $f(\underline{x})$ and every choice of m elements out of $\{x_1, \ldots, x_n\}$ are statistically independent. Then, f is correlation-immune of order m.
$\Leftarrow$ Let f be correlation-immune of order m. Then, there is

$$\begin{aligned} P(f(\underline{x}) = a \mid x_{i_1} = 0, \ldots, x_{i_m} = 0) = \\ P(f(\underline{x}) = a \mid x_{i_1} = 0, \ldots, x_{i_m} = 1) = \\ = \cdots = \\ P(f(\underline{x}) = a \mid x_{i_1} = 1, \ldots, x_{i_m} = 1) = \\ P(f(\underline{x}) = a). \end{aligned}$$

By this result, we can conclude that the value a appears in all 2^m $(n-m)-$dimensional cubes with the same frequency. □

In cryptographic applications, the mth order correlation-immune functions which are also balanced are of particular interest. For those, $K = 2^{n-m-1}$.

3 Construction of mth order correlation-immune functions

In the following, an algorithm for the construction of mth order correlation-immune functions with n binary input variables is presented. The algorithm is mainly based on theorem 2.1. If the binary value '1' assigned to the vertices of an $(n-m)$-cube, lying in an n-cube, appears with frequency K within this $(n-m)$-cube, we will say that the K-condition is fulfilled. If K is chosen to be $K = 2^{n-m-1}$, then the constructed function is also balanced. The application of algorithm 3.1 yields always one function with the desired properties. The assignments and the selections of $(n-m)$-cubes with minimum number of free vertices are controlled by the output of a random generator. This ensures that the output is able to construct distinct functions. Within the algorithm, parameter j describes the number of taken decisions if there are different possibilities of doing the assignments, all of them fulfilling the K-condition. To apply the algorithm, it is necessary to choose appropriate n, m and K.

Algorithm 3.1

1. $j := 1$
2. *select an $(n-m)$-cube, assign binary values to its vertices in a way fulfilling the K-condition, and note the assignment in a listing*
3. *if there exists an $(n-m)$-cube not considered before with no degrees of freedom to fulfill the K-condition for the assignment of the values to the vertices, then*
   - (a) *assignment of the binary values to the vertices fulfilling the K-condition in the considered $(n-m)$-cube*
   - (b) *if the K-condition is not fulfilled in all $(n-m)$-cubes, then*
     - i. *note assignment no. j to be bad and cancel the assignment no. j*
     - ii. $j := j-1$

     else
     - i. *note assignment no. j in the listing*
     - ii. *go to step 3*
4. *if there still exist vertices in the n-cube without assignment, then*
   - (a) *select an $(n-m)$-cube with minimum number of free vertices*
   - (b) *j:=j+1*
   - (c) *if there exists at least one possibility for the assignment to the vertices of the $(n-m)$-cube which is not noted in the listing to be bad, then*
     - i. *assignment of the binary values to the vertices fulfilling the K-condition in the $(n-m)$-cube in this way, as not noted in the listing to be bad*
     - ii. *note assignment no. j in the listing*
     - iii. *if the K-condition is not fulfilled in the other $(n-m)$-cubes, then*
       - A. *note assignment no. j to be bad and cancel the assignment no. j*
       - B. *go to step 4c*

else

A. go to step 3

else

i. j:=j-1

ii. note assignment no. j to be bad and cancel the assignment no. j

iii. delete the notes with attribute 'bad' and no. k for all k > j

iv. go to step 4c

else

end.

The result of the algorithm is an n-cube with all its vertices assigned a binary value. This can be easily translated into Boolean algebra or in the algebraic normal form (ANF) defined over $GF(2)$. In general, the ANF is preferred in literature.

Theorem 3.1 *Algorithm 3.1 is a method to construct all functions* $f : GF(2)^n \rightarrow GF(2), \underline{x} \rightarrow f(\underline{x})$, *having the property of correlation-immunity of order* m *with* $0 < m < n$.

Proof: By theorem 2.1, all mth order correlation-immune functions with n input variables satisfy the K-condition in all $(n-m)$–dimensional cubes with $K = 2^{n-m-1}$. Depending on the assignment in the first $(n-m)$–dimensional cube at the start of algorithm 3.1, all functions having the same vertex assignments in the first $(n-m)$–cube can be constructed obeying the K-condition in all further steps. This is valid for all combinatorial possibilities of the assignments in the first $(n-m)$–cube. Applying the method of algorithm 3.1, all these functions can be found fulfilling the K-condition in all $(n-m)$–dimensional cubes. By theorem 2.1, this yields all mth order correlation-immune functions. □

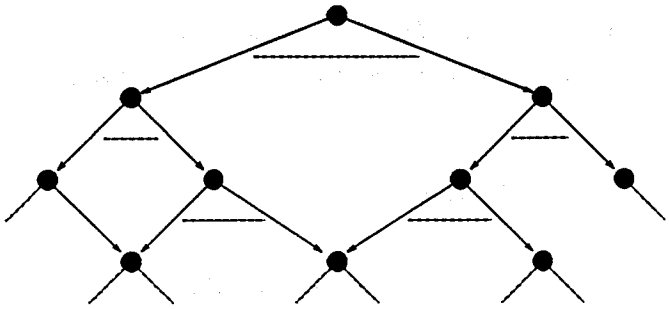

Fig. 1 : Structure of the graph representing the construction

The construction method given in algorithm 3.1 can be looked at in a graph theoretic way. Decisions with degrees of freedom are represented as branches; states within the construction, having alternative possibilities for the assignment of the binary values to the vertices of the $(n-m)$-cube, are the nodes. Only those phases in the n-cube during the construction are to be understood as states (nodes), which have degrees of freedom for further assignments. Nodes with no possibility of choosing an assignment to fulfill the K-condition have no branches. The graph induced by the construction method is directed and free of cycles. There are nodes which can be reached by more than one path. Fulfilling necessary conditions of mth order correlation-immune functions step by step in a depth first search manner, the application of algorithm 3.1 is more efficient than exhaustive search.

4 Complexity considerations

In this section, we will focus on the complexity aspects of the construction in algorithm 3.1. Here, our main interest is on two parameters. The first parameter is the maximum possible value of parameter j in algorithm 3.1 which leads to the maximum number of good decisions, that have to be taken to construct an mth order correlation-immune function with n input variables obeying the $K-$condition, where 'good' means the contrary of the term 'bad' which was used in algorithm 3.1. The second parameter is an upper bound of the frequencies to find $(n-m)-$cubes with certain numbers of free vertices in algorithm 3.1. With this parameter, the maximum number of branches at a node in the graph representation can be calculated. An exact mathematical description of algorithm 3.1 seems to be complicated. This is the reason for analyzing algorithm 3.1 in a modified manner. In the modification, the K-condition is dropped and the vertices of the n-cube are only marked instead of assigning binary values to them. The set of marked vertices will be called A.

Algorithm 4.1

1. $A = \{\}$
2. *select an $(n-m)$-cube and mark its vertices, marked vertices becoming elements of A*
3. *if not all vertices of the n-cube are elements of A, then*
   (a) *select an $(n-m)$-cube $\not\subseteq A$ with minimum number of vertices not in A, vertices of the $(n-m)$-cube becoming elements of A*
   (b) *go to 3*
4. *end*

The gradual marking of vertices of the n-cube done in algorithm 4.1 shows all the paths through the n-cube which are possible obeying only the minimality condition of algorithm 3.1. In general, with the application of the K-condition, algorithm 3.1 has stronger requirements. So, applying algorithm 3.1, there are no paths through the n-cube, which are not included in the set of all paths, applying algorithm 4.1. As already mentioned, the parameter j gives the number of decisions for the assignments applying algorithm 3.1. In algorithm 3.1, decisions are only possible if there are degrees of freedom, and these can only exist if the $(n-m)$-cube obeying the minimality condition has at least 2 vertices for assigning binary values. So all we have to do is counting the events, applying algorithm 4.1 when the set A is growing by at least 2 elements. This leads to an upper bound denoted by $H(n,m)$ for the parameter j in algorithm 3.1.

Now, for the sake of simplicity, we introduce an expression for the set of vertices of an $n-$cube, which are inducing a $k-$cube with $0 < k < n$. Consider a $k-$cube, given by the index set $I = \{i_1 = 1, \ldots, i_k = k\}$. This $k-$cube is identified by the set $\{(x_1, \ldots, x_n) | (x_j \in GF(2), \text{if } j \in I) \text{ and } (x_j = a_j, \text{ with } a_j \in GF(2) \text{ fixed, if } j \notin I)\}$. In the further considerations, this $k-$cube will be described by $(x_1 \ldots, x_k, a_{k+1}, \ldots, a_n)$.

Theorem 4.1 *Let $1 \leq j \leq n$ and let the index set $I = \{i_1, \ldots, i_j\} \subset \{1, \ldots, n\}$ induce a $j-$cube $A = \{(x_1, \ldots, x_n) \mid (x_i \in GF(2), \text{ if } i \in I) \text{ and } (x_i = a_i, \text{ with } a_i \in GF(2) \text{ fixed, if } i \notin I)\}$. Furthermore, let $1 \leq k \leq j$. Given the index sets I_1, I_2 with k elements, with $I_1 \not\subseteq I$, $I_2 \not\subseteq I$, $I_1 \cap I \neq \{\}$, $I_2 \cap I \neq \{\}$ and $|I_1 \cap I| = |I_2 \cap I| + 1$, which are inducing the $k-$cubes A_1 and A_2. Then, there is $|A \cup A_1| < |A \cup A_2|$.*

Proof: Let $k = k' + k''$ with $k', k'' > 0$. Consider the following cubes, without restriction of generality.

$$\begin{array}{ll} j-\text{cube } A & (x_1, \ldots, x_j, a_{j+1}, \ldots, a_n) \\ k-\text{cube } A_1 & (a_1, \ldots, a_{j-k''-1}, x_{j-k''}, \ldots, x_{j+k'-1}, a_{j+k'}, \ldots, a_n) \\ k-\text{cube } A_2 & (a_1, \ldots, a_{j-k''}, x_{j-k''+1}, \ldots, x_{j+k'}, a_{j+k'+1}, \ldots, a_n) \end{array}$$

We have $|A| = 2^j$ and $|A_1| = |A_2| = 2^k$. The $k-$cubes A_1 and A_2 can be obtained by the union of $(k''+1)$-cubes, respectively k''-cubes:

$$\begin{aligned} A_1 &= (a_1, \ldots, a_{j-k''-1}, x_{j-k''}, \ldots, x_{j+k'-1}, a_{j+k'}, \ldots, a_n) \\ &= (a_1, \ldots, a_{j-k''-1}, x_{j-k''}, \ldots, x_j, a_{j+1}, \ldots, a_{j+k'-1}, a_{j+k'}, \ldots, a_n) \\ &\qquad \cup \\ &\quad (a_1, \ldots, a_{j-k''-1}, x_{j-k''}, \ldots, x_j, a_{j+1}, \ldots, a_{j+k'-1}+1, a_{j+k'}, \ldots, a_n) \\ &\qquad \cup \\ &\qquad \ldots \\ &\qquad \cup \\ &\quad (a_1, \ldots, a_{j-k''-1}, x_{j-k''}, \ldots, x_j, a_{j+1}+1, \ldots, a_{j+k'-1}+1, a_{j+k'}, \ldots, a_n) \end{aligned}$$

The first $(k''+1)-$cube on the right side in the preceding equation is completely contained in A. All others $(k''+1)-$cubes have no elements in common with A. A $(k''+1)-$cube has $2^{k''+1}$ elements. Hence, there is $|A \cup A_1| = 2^j + 2^k - 2^{k''+1}$.

$$\begin{aligned} A_2 &= (a_1, \ldots, a_{j-k''}, x_{j-k''+1}, \ldots, x_{j+k'}, a_{j+k'+1}, \ldots, a_n) \\ &= (a_1, \ldots, a_{j-k''}, x_{j-k''+1}, \ldots, x_j, a_{j+1}, \ldots, a_{j+k'}, a_{j+k'+1}, \ldots, a_n) \\ &\qquad \cup \\ &\quad (a_1, \ldots, a_{j-k''}, x_{j-k''+1}, \ldots, x_j, a_{j+1}, \ldots, a_{j+k'}+1, a_{j+k'+1}, \ldots, a_n) \\ &\qquad \cup \\ &\qquad \ldots \\ &\qquad \cup \\ &\quad (a_1, \ldots, a_{j-k''}, x_{j-k''+1}, \ldots, x_j, a_{j+1}+1, \ldots, a_{j+k'-1}+1, a_{j+k'}, \ldots, a_n) \end{aligned}$$

The first $k''-$cube on the right side in the preceding equation is completely contained in A. All others $k''-$cubes have no elements in common with A. A $k''-$cube has $2^{k''}$ elements. Hence, there is $|A \cup A_2| = 2^j + 2^k - 2^{k''}$. Comparing the number of elements $|A \cup A_1| = 2^j + 2^k - 2^{k''+1} < |A \cup A_2| = 2^j + 2^k - 2^{k''}$, we have the desired result. □

Theorem 4.2 *Let the vertices of an $n-$cube contained in the set A induce a $j-$cube with $k \leq j < n$ and $k = n - m$ by algorithm 4.1. By the minimality condition of algorithm 4.1, a $k-$cube is selected, and A gets new elements. Then the smallest cube, containing all elements of A, has dimension $j+1$, and the number of new elements in the set A is 2^{k-1}.*

Proof: Without restricting the generality, consider the $j-$cube $A = (x_1, \ldots, x_j, a_{j+1}, \ldots, a_n)$. By theorem 4.1 it is known that the smallest cube containing the union of A and a k-cube given by the minimality condition of algorithm 4.1 is a $(j+1)$-cube, for example $(x_1, \ldots, x_{j+1}, a_{j+2}, \ldots, a_n)$. This $(j+1)-$cube can be splitted in disjunct $j-$cubes A and $A' = (x_1, \ldots, x_j, a_{j+1}+1, \ldots, a_n)$. By theorem (4.1), the number of new elements coming to A is smallest, if the

$k-$cube is chosen in such a way, that just one of the fixed $(n-j)$ components of the $j-$cube becomes variable. Without restricting the generality, we can do the following choice for $k-$cube: $(a_1,\ldots,a_{j+1-k},x_{j+2-k},\ldots,x_j,x_{j+1},a_{j+2},\ldots,a_n)$. This $k-$cube can be splitted in two $(k-1)-$cubes each having 2^{k-1} vertices:

$$\begin{array}{c}(a_1,\ldots,a_{j+1-k},x_{j+2-k},\ldots,x_j,a_{j+1},\ldots,a_n)\\ \bigcup\\ (a_1,\ldots,a_{j+1-k},x_{j+2-k},\ldots,x_j,a_{j+1}+1,\ldots,a_n).\end{array} \tag{2}$$

The upper $(k-1)-$cube in (2) is completely contained in A. The lower $(k-1)$-cube in (2) and A have no elements in common. So, A gets 2^{k-1} new elements by selecting this $k-$cube. Such a $k-$cube exists for all j with $k \leq j < n$. □

Theorem 4.3 *Consider a $(j+1)-$cube with $k \leq j < n$ in the context of algorithm 4.1, having one half completely in A, while the other half is not completely in A. Then there exists always at least one $k-$cube which brings not more than $2^{k-1}-1$ new elements to A.*

Proof: Given a j-cube and a $(j+1)$-cube: $(x_1,\ldots,x_j,a_{j+1},\ldots,a_n) \subset A \subset (x_1,\ldots,x_j,x_{j+1},a_{j+2},\ldots,a_n)$. Consider the vertex $(a_1,\ldots,a_j,a_{j+1}+1,a_{j+2},\ldots,a_n)$ which is supposed to be in A, but not contained in the $j-$cube. There exists always a $k-$cube, containing this vertex and at least 2^{k-1} further elements, which are already contained in A: $(a_1,\ldots,a_{j-k+1},x_{j-k+2},\ldots,x_{j+1},a_{j+2},\ldots,a_n)$. Therefore, the number of vertices becoming elements of A can not be greater than $2^{k-1}-1$. This method can be applied to all k-cubes not completely contained in A until all elements of the $(j+1)-$cube are contained in A. □

Theorem 4.4 *Given an $n-$cube whose vertices are processed by the instructions of algorithm 4.1. Within the application of algorithm 4.1, the $n-$cube runs always beside others through these states in which the number of the elements of set A equals to a power 2^j with $k \leq j \leq n$. Furthermore, these 2^j elements of set A always induce an $n-$cube.*

Proof: The proof follows by induction referring to theorems 4.2 and 4.3.

$j = k$: The number of elements in A is 2^k. These elements are the vertices of a $k-$cube.

$j \rightarrow j+1$ with $j < n$: Let A contain 2^j elements inducing a $j-$cube. The application of algorithm 4.1 requires the selection of a $k-$cube described by theorem 4.2, where the number of new elements in A is 2^{k-1}. In the further steps, the application of algorithm 4.1 is described by theorem 4.3 until A induces a $(j+1)-$cube. Then A contains 2^{j+1} elements. □

With the results of theorems 4.2 - 4.4, we have described the possible paths through the $n-$cube in algorithm 4.1. Theorem 4.4 says that if A contains 2^{n-1} elements, then these induce an $(n-1)-$cube.

In the following, we will see how the consideration of the interesting complexity parameters concerning the application of algorithm 4.1 with given n and m can be decomposed into double application of algorithm 4.1 with $n' = n-1$, $m' = m-1$, and $n'' = n-1$, $m'' = m-1$, respectively.

Theorem 4.5 *Given $1 < m < n$. Let an $n-$cube be processed by the instructions of algorithm 4.1, while adding vertices of $k-$dimensional cubes to A. Furthermore, let $k = n-m$. If the*

application of algorithm 4.1 is considered as long as $|A| < 2^{n-1}$, then this has the same result as application of algorithm 4.1 to an $n'-$cube with $n' = n-1$ and $m' = m-1$.

Proof: By the result of theorem 4.4 it is known that the smallest cube containing A as long as $|A| < 2^{n-1}$ is a $(n-1)$-cube. Within algorithm 4.1 with given n and m, vertices of a $k-$cube with $k = n-m$ become elements of A. As long as $|A| < 2^{n-1}$, all the k-cubes lie completely within the $(n-1)$-cube. This is equivalent to applying the algorithm 4.1 with n' and m'. With $k = n-m$ and $k = n'-m'$, we have $n-m = n'-m'$, and because of $n' = n-1$, there is $m' = m-1$. □

We conclude that with the notation of theorem 4.5 the application of algorithm 4.1 with n, m as long as $|A| < 2^{n-1}$ and its application with n', m' yield the same maximal number of events when the set A is growing by at least 2 elements. Furthermore, also the frequencies of adding specific numbers of elements to A are equal. In the next two theorems, it will be shown that there exist similar properties concerning the application of algorithm 4.1 with n, m when $2^{n-1} \leq |A| < 2^n$ and the application with $n'' = n-1$, $m'' = m$.

Theorem 4.6 *Given $0 < m < n-1$. Let an $n-$cube be processed by the instructions of algorithm 4.1 and let $2^{n-1} < |A| < 2^n$. Let A' be an $(n-1)-$cube completely contained in A. If there exists a $k-$cube Ψ', with $k = n-m$ and $A' \cap \Psi' = \{\}$, fulfilling the minimality condition while adding its elements to A, then there is always a $k-$cube Ψ with $|A' \cap \Psi| = 2^{k-1}$, also fulfilling the minimality condition.*

Proof: By assumption, A consists of a $(n-1)$-cube A' and further vertices, which are combined in a set Θ. There is $\Theta \cap A' = \{\}$, and by theorem 4.2 we know, that $|\Theta| \geq 2^{k-1}$. Then, the number of new elements coming to A applying algorithm 4.1 is always smaller than 2^{k-1}, by the result of theorem 4.3. There is $A = A' \cup \Theta \neq (x_1, \ldots, x_n)$. Suppose $A' = (x_1, \ldots, x_{n-1}, a_n)$ and $\Psi' = (a_1, \ldots, a_{n-k-1}, x_{n-k}, \ldots, x_{n-1}, a_n + 1)$, which are disjunct. Furthermore, suppose that the new elements of A resulting by adding vertices of Ψ' to A lie in a $(k-1)$-cube, given by $(a_1, \ldots, a_{n-k-1}, a_{n-k}, x_{n-k+1}, \ldots, x_{n-1}, a_n + 1)$. All other elements of Ψ' lie completely in Θ. A $k-$cube Ψ, with $\Psi \cap \Psi' = (a_1, \ldots, a_{n-k-1}, a_{n-k}, x_{n-k+1}, \ldots, x_{n-1}, a_n + 1)$ and half of its elements completely contained in A', can always be found with $\Psi = (a_1, \ldots, a_{n-k}, x_{n-k+1}, \ldots, x_n)$. If the growth of A by adding the elements of Ψ' is minimal, then also the growth by adding the elements of Ψ is minimal. □

Theorem 4.7 *Given $0 < m < n-1$. Let an $n-$cube be processed by the instructions of algorithm 4.1, while adding vertices of $k-$dimensional cubes to A. Furthermore, let $k = n-m$. If the application of algorithm 4.1 is considered from the step, when $2^{n-1} \leq |A| < 2^n$, then the result is the same as application of algorithm 4.1 to an $n''-$cube with $n'' = n-1$ and $m'' = m$.*

Proof: By theorem 4.4 we conclude that the vertices not contained in A induce an $(n-1)-$cube, if A contains exactly 2^{n-1} elements. By theorems 4.2, 4.3 and 4.6 we know that there always exist such k-cubes fulfilling the minimality condition of algorithm 4.1 and having at least one half in a $(n-1)$-cube which is completely contained in A. Each half of this k-cube can be considered as a $(k-1)-$cube. With $k-1 = n''-m''$, $k = n-m$ and $n'' = n-1$, we have $m'' = n''-k+1 = n-1-k+1 = n-k = m$. □

This allows to consider H as a function of n and m and to construct H recursively for $1 < m < n-1$:

$$H(n,m) = H(n-1,m-1) + H(n-1,m). \tag{3}$$

The recursion can only be applied when $m < n-1$ and $m > 1$. Therefore, $H(n,m)$ has to be calculated separately for the cases $m = 1$ and $m = n-1$.

Theorem 4.8 *Let $n \geq 2$ and $m = n-1$. Then application of algorithm 4.1 to an $n-$cube results in $H(n, m = n-1) = 1$.*

Proof: In the case $n \geq 2$ and $m = n-1$, the k-cubes have dimension $k = n-m = n-n+1 = 1$. An 1-cube has 2 vertices. Choosing an 1−cube at the beginning of algorithm 4.1, then, by the minimality condition, all further 1-cubes have only 1 vertex not contained in A. Because $H(n,m)$ is the number of events when the set A is growing by at least 2 elements, we have $H(n, m = n-1) = 1$. □

Theorem 4.9 *Let $n \geq 2$ and $m = 1$. Then application of algorithm 4.1 to an $n-$cube results in $H(n, m = 1) = n-1$.*

Proof: Without restriction of generality, consider the following application of algorithm 4.1:

step	cube
1. $(n-1)-$ cube	$(x_1, \ldots, x_{n-1}, a_n)$
2. $(n-1)-$ cube	$(x_1, \ldots, x_{n-2}, a_{n-1}, x_n)$
3. $(n-1)-$ cube	$(x_1, \ldots, x_{n-3}, a_{n-2}, x_{n-1}, x_n)$
.	.
.	.
(n-1). $(n-1)-$ cube	$(x_1, a_2, x_3, \ldots, x_n)$

Now, there are still 2 vertices of the $n-$cube not contained in A: $P_1 = (a_1, a_2+1, \ldots, a_n+1)$ and $P_2 = (a_1+1, a_2+1, \ldots, a_n+1)$. Applying the minimality condition of algorithm 4.1, both vertices become elements of A in two further steps. In each step, there is one element added to A. Because $H(n,m)$ is the number of events when the set A is growing by at least 2 elements, we have $H(n, m = n-1) = n-1$. □

With these results, a general description for $H(n,m)$ is possible.

Theorem 4.10 *Let $n \geq 2$ and $0 < m < n$. Then application of algorithm 4.1 to an $n-$cube results in:*

$$H(n,m) = \binom{n-1}{m}. \tag{4}$$

Proof: In the proof, we consider three cases. These are indicated by α, β and γ in the following figure.

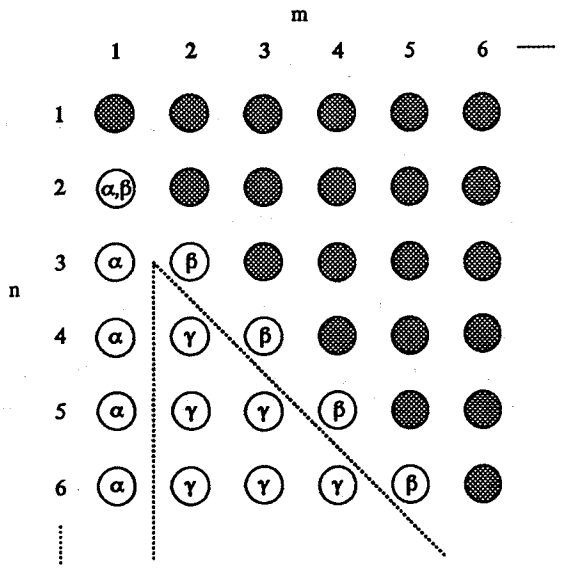

Fig. 2 : Cases α, β and γ

α): $n \geq 2$ and $m = 1$

$$H(n, m = 1) = \binom{n-1}{1} = n - 1$$

This coincides with the result of theorem 4.9.

β): $n \geq 2$ and $m = n - 1$

$$H(n, m = n - 1) = \binom{n-1}{n-1} = 1$$

This coincides with the result of theorem 4.8.

γ): Let $m > 1$ and let $n \geq 4$ and $n > m + 1$. Applying recursion (3) an appropriate number of times, which is only valid for the conditions of γ, we obtain cases that match the conditions of α and β. In these cases, the recursion can not be applied anymore. But there, the correctness of the assumption is already shown. The proof is given by induction. Because it is already shown that the assumption holds for some initial values (see α and β), all we have to do is to verify that the step from n to $n + 1$ holds.

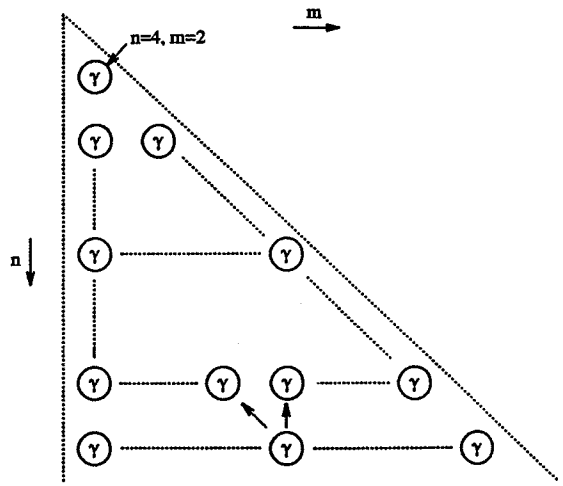

Fig. 3 : Application of the recursion for $H(n,m)$

Let $H(n,m) = \binom{n-1}{m}$ and consider $H(n+1,m)$:

$$\begin{aligned} H(n+1,m) &= H(n,m-1) + H(n,m) = \\ &= \binom{n-1}{m-1} + \binom{n-1}{m} = \\ &= \binom{n}{m} \end{aligned}$$

This is the desired result. □

Definition 4.1 *Let $0 < m < n$ and let $c > 0$. The function $\#(c,n,m)$ describes how often c elements are added to A when applying algorithm 4.1 with n and m.*

It is obvious that with given $0 < m < n$ and $k = n - m$, $\#(c,n,m) = 0$, if $c < 1$ or $c > 2^k$. Therefore, if $\#(c,n,m) \neq 0$, then $1 \leq c \leq 2^k$. Similar to theorem 4.10, we can establish a recursion for $\#(c,n,m)$.

Theorem 4.11 *Let $n \geq 2$ and $0 < m < n$. Then application of algorithm 4.1 to an $n-$cube results in:*

$$\#(c,n,m) = \#(c,n-1,m-1) + \#(c,n-1,m). \tag{5}$$

Proof: The proof follows by theorems 4.1 - 4.7. □

Theorem 4.12 *Let $2 \leq n$ and $m = 1$. Applying algorithm 4.1 to an $n-$cube, there will be the following frequencies $\#(c,n,m=1)$ with $1 \leq c \leq 2^{n-1}$:*

$$\#(c,n,m=1) = \begin{cases} 1, & \text{if } c = 2^i \text{ with } i = 1 \ldots n-1, \\ 2, & \text{if } c = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Proof: Consider the following application of algorithm 4.1:

step	cube	growth of A
1. $(n-1)$– cube	$(x_1, \ldots, x_{n-1}, a_n)$	2^{n-1}
2. $(n-1)$– cube	$(x_1, \ldots, x_{n-2}, a_{n-1}, x_n)$	2^{n-2}
3. $(n-1)$– cube	$(x_1, \ldots, x_{n-3}, a_{n-2}, x_{n-1}, x_n)$	2^{n-3}
.	.	.
.	.	.
(n-1). $(n-1)$– cube	$(x_1, a_2, x_3, \ldots, x_n)$	2

Now, there are still two vertices of the n-cube not contained in A: $P_1 = (a_1, a_2+1, \ldots, a_n+1)$ and $P_2 = (a_1+1, a_2+1, \ldots, a_n+1)$. These vertices become elements of A in two further steps where the growth of A is 1. The desired result follows.□

Theorem 4.13 *Let $2 \leq n$ and $m = n-1$. Applying algorithm 4.1 to an $n-$cube, there will be the following frequencies $\#(c, n, m = n-1)$ with $1 \leq c \leq 2^{n-1}$:*

$$\#(c, n, m = n-1) = \begin{cases} 1, \text{ if } c = 2, \\ 2^n - 2, \text{ if } c = 1, \\ 0, \text{ otherwise.} \end{cases}$$

Proof: Applying algorithm 4.1 with $m = n-1$, we have to consider the vertices of 1-cubes. At the start of the algorithm, there are two vertices becoming elements of A. In all further steps, the growth of A is only 1. Therefore, we have $\#(c = 2, n, m = n-1) = 1$ and $\#(c = 1, n, m = n-1) = 2^n - 2$. For all other values of c, there is $\#(c, n, m = n-1) = 0$. This is the desired result. □

Theorem 4.14 *Let $n > 2$ and $1 < m < n$. Considering $c = 2^{n-m}$, applying algorithm 4.1 yields $\#(c = 2^{n-m}, n, m) = 1$.*

Proof: At the beginning of algorithm 4.1, the growth of A is given by 2^{n-m} elements. In all further steps, there are only such values c, which are $c = 2^{n-m-1}$ by theorem 4.2 or $c < 2^{n-m-1}$ by theorem 4.3. Therefore, $\#(c = 2^{n-m}, n, m) = 1$. □

Theorem 4.15 *Let $2 \leq n$ and $1 \leq m \leq n-1$. Then $\#(c, n, m) \neq 0$ only for those c, if $c = 2^j$ with $j = 0, 1, \ldots, (n-m)$, otherwise $\#(c, n, m) = 0$.*

Proof: By theorems 4.5 and 4.7, applying algorithm 4.1 with given n and m can be considered in a manner of twice applying algorithm 4.1 with $n' = n-1$, $m' = m-1$ and $n'' = n-1$, $m'' = m$. This can be repeated until the conditions of theorem 4.12 or theorem 4.13 are fulfilled. By these theorems, there are only such growth values c, that equal 2^j with $j = 0, 1, \ldots, (n-m)$. □

With these results, we can give a general description for $\#(c = 2^j, n, m)$, which is presented in the next theorem.

Theorem 4.16 *Let $n \geq 2$ and $0 < m < n$. Furthermore, let $j \in \{1, 2, \ldots, (n-m)\}$. Applying algorithm 4.1 then yields the following frequencies for adding $c = 2^j$ elements to A:*

$$\#(2^j, n, m) = \binom{n-j-1}{m-1} \tag{6}$$

Proof: Like in the proof of 4.10, we consider 3 cases. (See figures in the proof of theorem 4.10.)

α): $n \geq 2$ and $m = 1$
In this case, $1 \leq j \leq n - m = n - 1$. For all those j, we have:

$$\#(2^j, n, m = 1) = \binom{n-j-1}{m-1} = \binom{n-j-1}{0} = 1$$

This coincides with the result of theorem 4.12.

β): $n \geq 2$ und $m = n - 1$
Here, only $j = 1$ is relevant, because of $n - m = 1$. For $j = 1$ we have:

$$\#(2, n, m = n - 1) = \binom{n-j-1}{m-1} = \binom{n-2}{n-2} = 1$$

This coincides with the result of theorem 4.13.

γ): Let $m > 1$ and let $n \geq 4$ and $n > m - 1$. Therefore, $j \in \{1, 2, \ldots, (n - m)\}$. Recursion (5) can only be applied for case γ. Applying this recursion an appropriate number of times, we obtain case α and β. The proof is given by induction. The start values are given by cases α and β. It remains to proof the step from n to $n + 1$.

Let $\#(c = 2^j, n, m) = \binom{n-j-1}{m-1}$ and consider $\#(c = 2^j, n + 1, m)$.

$$\begin{aligned}
\#(c = 2^j, n+1, m) &= \#(c, n, m) + \#(c, n, m-1) = \\
&= \tbinom{n-j-1}{m-2} + \tbinom{n-j-1}{m-1} = \\
&= \tfrac{(n-j-1)!}{(m-2)!(n-j-m+1)!} + \tfrac{(n-j-1)!}{(m-1)!(n-j-m)!} = \\
&= \tfrac{(n-j-1)!(m-1)!+(n-j-1)!(n-j-m+1)!}{(m-1)!(n-j-m+1)!} = \\
&= \tfrac{(n-j)!}{(m-1)!(n-j-m+1)!} = \\
&= \tbinom{n-j}{m-1}
\end{aligned}$$

□

5 New Upper Bounds

With the result of theorem 4.16, we know how often $c = 2^j$ binary elements are assigned to the vertices of the $n-$cube within the construction of balanced and mth order correlation-immune functions while applying algorithm 3.1 to the $n-$cube. Each $k-$cube, $k = n - m$, has 2^{k-1} '0's and 2^{k-1} '1's. The number of possibilities is biggest if the $c = 2^j$ free vertices of a $k-$cube were assigned 2^{j-1} '0's and 2^{j-1} '1's. This yields an upper bound for the number of balanced and mth order correlation-immune functions which is summarized in the following theorem.

Theorem 5.1 *Let $1 \leq m < n$. The number of balanced and mth order correlation-immune functions with n inputs is upperbounded by:*

$$\prod_{j=1}^{n-m} \binom{2^j}{2^{j-1}}^{\binom{n-j-1}{m-1}}. \tag{7}$$

It has to be noted that this bound is valid for all m and n with $0 < m < n$. This result can be compared to the upper bound presented by Yang and Guo (see figure 4) which gives the number of 1st order ($m = 1$) correlation-immune functions [YaGu 95]:

$$\sum_{k=0}^{2^{n-1}} \sum_{r=0}^{k} \binom{2^{n-2}}{r}^2 \binom{2^{n-2}}{k-r}^2. \tag{8}$$

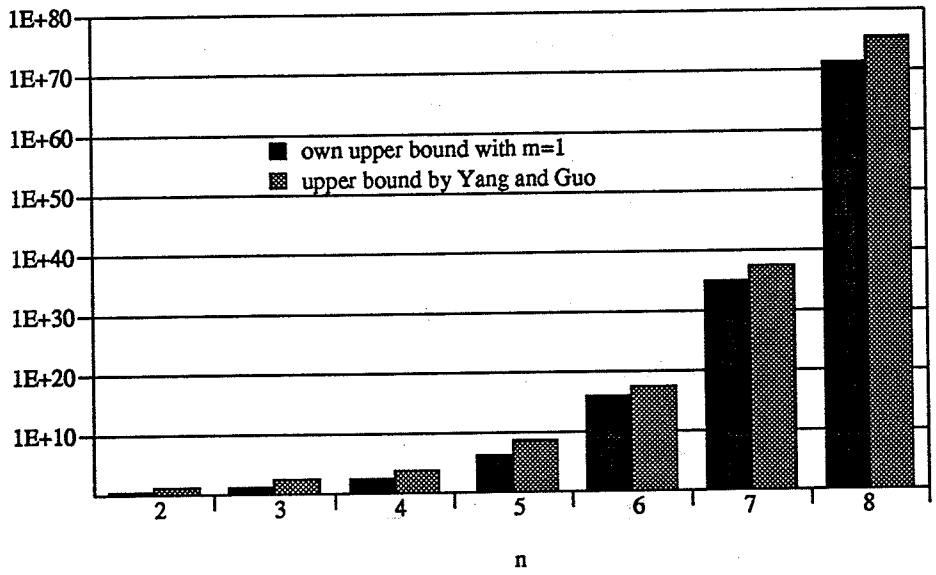

Fig. 4 : Comparison of upper bounds

6 Acknowledgements

I am grateful to Professor Firoz Kaderali and Professor Werner Poguntke for the supervision of my work. Furthermore, I would like to thank Professor Ulrich Faigle and Professor Walter Kern (both University Twente) for helpful discussions about some contents of this paper.

References

[CCCS 91] P. Camion, C. Carlet, P. Charpin, N. Sendrier: *'On Correlation-immune Functions'*, Advances in Cryptology: Crypto '91, Proceedings, Lecture Notes on Computer Science 576, 1991, p. 86-100

[GuMa 88] Xiao Guo-Zhen, James L. Massey: *'A Spectral Characterization of Correlation-Immune Combining Functions'*, IEEE Transactions on Information Theory, Vol. 34, No. 3, May 1988, p. 569-571

[RuSt 85] Rainer A. Rueppel, Othmar J. Staffelbach: *'Products of Linear Recurring Sequences with Maximum Complexity'*, IEEE Transactions on Information Theory, Vol. IT-33, No. 1, Jan. 1987, p. 124-131

[SeZh 93] Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng: *'On Constructions and Nonlinearity of Correlation Immune Functions (Extended Abstract)'*, Advances in Cryptology: Eurocrypt '93, Proceedings, Lecture Notes on Computer Science, 765, 1993, p. 181-199

[Sieg 84] Thomas Siegenthaler: *'Correlation-Immunity of Nonlinear Combining Functions of Cryptographic Applications'*, IEEE Transactions on Information Theory, Vol. IT-30, No. 5, Sep. 1984, p. 776-780

[Sieg 85] Thomas Siegenthaler: *'Decrypting a Class of Stream Ciphers Using Ciphertext Only'*, IEEE Transactions on Computers, Vol. C-34, No. 1, Jan. 1985, p.81-85

[YaGu 95] Yi Xian Yang, Baoan Guo: *'Further Enumerating Boolean Functions of Cryptographic Significance'*, Journal of Cryptology, 1995, 8: p.115-122

Efficient Stream Cipher with Variable Internal State

André Zúquete and Paulo Guedes

IST / INESC
R. Alves Redol 9, 1000 Lisboa, Portugal
email: (Andre.Zuquete, Paulo.Guedes)@inesc.pt

Abstract

This paper presents an efficient stream cipher using an internal state with variable structure and evolution. Arbitrarily large internal states can be used in order to defeat brute-force guessing attacks without compromising the performance of cipher, and possibly improving it. Attacking is made even more complicated by dynamically chosing different topologies and evolutions for the cipher's internal state.

The cipher controls the evolution of its internal state by using both an external keyed pseudo-random generator (EKPRG), either cryptographically strong or weak, and plaintext feedback. The plaintext feedback reduces the probability of producing cyclic keystreams without compromising the security of the cipher.

The parameters controlling the structure and evolution of the cipher's internal state can be chosen in order to achieve different levels of security, memory consumption and performance. In terms of security, we evaluate the impact of this parameters in the strength of the cipher against brute-force guessing attacks. Concerning performance, we evaluate the encryption speed of the cipher using two different EKPRGs – 8-bit ARC4 and DES working in 8-bit OFB – with several topologies and evolutions of the internal state assuring very high security levels. When comparing against the ciphers used as external EKPRGs, we obtain a minimum encryption speedup of 7 % and 428 %, respectively, and a maximum speedup of 63 % and 2556 %.

1 Introduction

Traditional ciphers are based on fixed internal structures managed by fixed- or variable-length keys [1, 18]. The knowledge of the cipher's internal structure facilitates the development of attacks attempting to guess its contents or the key controlling them [4, 5, 6, 3, 15, 14, 20]. This problem can be reduced if ciphers support some degrees of freedom regarding their operational parameters, besides the size of the key, like the block cipher RC5 [17, 16].

In this paper we present a new efficient stream cipher allowing to dynamically choose the structure and evolution of its internal state. By using a flexible internal structure and a flexible algorithm to modify it, we introduce extra factors of complexity in the development of sophisticated attacks against specific implementations of the cipher.

The parameters controlling the structure and evolution of the cipher's internal state can be chosen in order to achieve different levels of security, memory consumption and and performance. Arbitrarily large internal states can be used in order to defeat brute-force guessing attacks without compromising the performance of cipher, and even improving it in some cases.

The cipher works like any other stream cipher: plaintext bytes are XORed with a keystream, producing the ciphertext, and the ciphertext is XORed with the same keystream to recover the original plaintext bytes. However, unlike most stream ciphers, our cipher uses both an external keyed pseudo-random generator (EKPRG) and plaintext feedback as sources of data for ran-

domising its internal state. The cipher's key (K) is used to set up the initial value of its internal state and also the key used by the EKPRG (see figure 1).

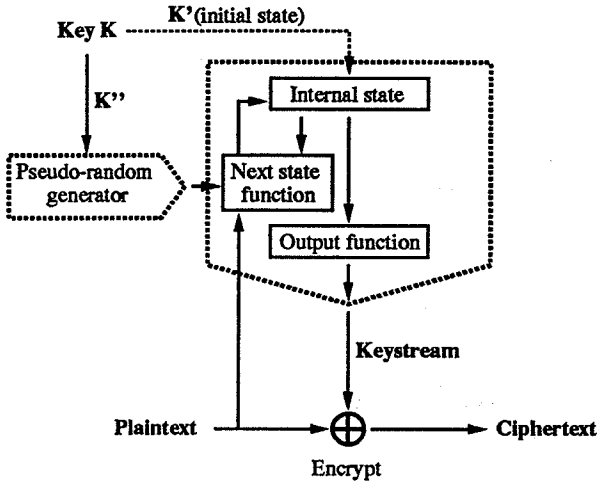

Figure 1: *Overview of the keystream generator.*

The EKPRG can be either cryptographically strong or weak, depending on security considerations. It can be a simple iterator over the bytes of a (possibly large) key, or a cryptographically strong byte generator like RC4 [18]. The choice of the right EKPRG is conditioned by the relevance of the data it provides to the overall security of the cipher, which depends on several other factors (e.g. the dimension of its internal structure). If the knowledge of that data is useless for attackers, then simpler and faster EKPRGs can be used.

The cipher produces blocks of keystream bytes from an arbitrarily large internal state, and uses the EKPRG to shuffle and modify it whenever new blocks are needed. The structure and evolution of its internal state is determined by a set of *control parameters*. Exhaustive search attacks against the cipher's internal state can be prevented to a great extent by keeping secret, or variable within known limited ranges, the value of the control parameters. For instance, we can derive the control parameters dynamically from the cipher's key K.

The evolution of the cipher's internal state, which is ruled by the control parameters, depends on its own contents, on values provided by the EKPRG, and on pseudo-random samples of plaintext bytes. The plaintext feedback, which is a reasonably random source of information, drastically reduces the probability of producing cyclic keystreams. The plaintext feedback is used in such a way that attackers controlling it (e.g. using chosen-plaintext attacks) cannot deterministically influence the keystream, or even conclude which keystream bytes were affected by plaintext bytes.

The performance of the cipher depends on the control parameters used to define and manage its internal state, and on the speed of the EKPRG. We evaluated the encryption speed of the cipher, when encrypting 10 Kbyte buffers, using two different (cryptographically strong) EKPRGs – 8-bit ARC4, a stream cipher allegedly compatible with the secret RC4 cipher, and DES working in 8-bit OFB – and several sets of control parameters assuring very high security levels. When comparing against the ciphers used as EKPRGs, we obtained a minimum encryption speedup of 7 % and 428 %, respectively, and a maximum speedup of 63 % and 2556 %.

The rest of this paper is organised as follows. The next section describes the cipher's keystream generator. Section 3 analyses the impact of the cipher's control parameters in its strength against known-plaintext attacks. Section 4 presents the implementation of the cipher and the evaluation of its performance and security. Section 5 presents the related work. Finally, in section 6 we present the conclusions.

2 The keystream generator

The keystream generator was designed using a system-theoretic approach in order to have a flexible and arbitrarily large internal state capable of defeating exhaustive search attacks. Such internal state is efficiently shuffled by an EKPRG (see Figure 1). This EKPRG can either be any generator typically used as a Vernam cipher, like generators using linear feedback shift registers, block ciphers running in OFB mode, RC4 [18], or any other generator cryptographically not strong enough to be used directly as a

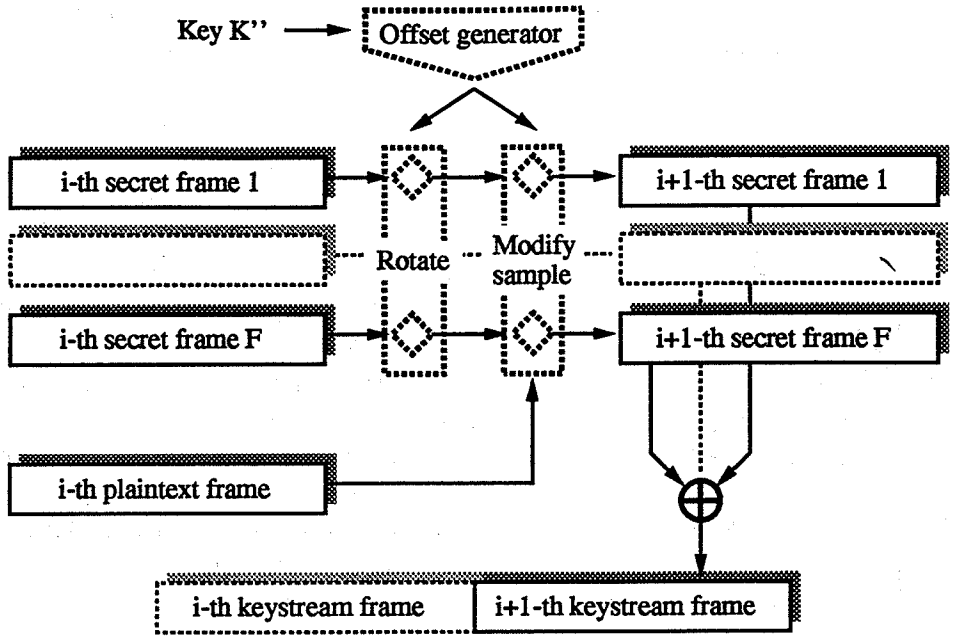

Figure 2: *Algorithm to generate a new keystream frame.*

Vernan cipher, like the Gifford's cipher [18, 8].

The generator efficiently produces pseudo-random blocks of keystream bytes from its internal state, and uses the EKPRG to shuffle it whenever new blocks are needed (see Figure 2). Samples of plaintext bytes are also used to further randomise it. The generator's internal state is independent from the internal state of the EKPRG.

The configuration and evolution of the generator's internal state, defined by control parameters, can be tailored to achieve different levels of security, performance, and memory consumption. If the control parameters are kept secret, then the implementation of the generator can be relaxed in order to use faster but cryptographically weaker EKPRGs. The generator's key K is used to compute two other keys that influence its overall behaviour – the initial value of the generator's internal state (K') and the key of its EKPRG (K'').

2.1 Algorithm overview

The keystream generation algorithm is presented in Figure 2 and works as follows. The keystream is generated frame by frame, and each frame has L bytes; these are used only once to encrypt or decrypt data. Each frame is computed from F L-byte long *secret frames* – SF^1, $\cdots$, SF^F, and each bit of a keystream frame depends on F bits, one from each secret frame. The initial value of secret frames, K', is discussed in §2.2.

To generate a new keystream frame, secret frames are first rotated, then an M-byte long sample of them is modified and, finally, they are combined (XORed) to produce a new keystream frame. Both rotation and modification operations are based on offsets provided by the EKPRG – the *offset generator*. The modification of secret frames makes them evolve pseudo-randomly each time a new keystream frame is generated. The rotation of secret frames should make consecutive keystream frames look very different, even with few modifications of secret frames (see §2.3). The modification of secret frames allows them to take, all together, any of the 2^{8LF} possible values, being thus capable of generating any of the 2^{8L} possible values for keystream frames.

The control parameters of the generator are the number (F) and length (L) of secret frames and the number of modifications per secret

frame (M). Exhaustive search attacks against the generator can be prevented to a great extent by keeping secret, or variable within known limited ranges, the value of control parameters used in particular instantiations of the generator. All these parameters can be dynamically computed from the key K and cannot be derived by inspecting the ciphertext.

2.2 Management of K, K' and K''

The secrecy of the initial contents of secret frames, K', is crucial to the overall security of the generator. The value of K' should be as secret and not structured as possible (the more random the better) to complicate guessing secret frames from keystream frames. Each $< K', K'' >$ pair should be fresh, i.e. never been used before, to minimise the probability of getting equal keystream sequences. Furthermore, if the pair $< K', K'' >$ is fresh but K' was already used, then we can use that pair provided that we start encrypting data with the second keystream frame, instead of the initial one (directly computed from K').

A possibility for computing a $< K', K'' >$ pair from a fresh K, is to generate K' using directly the output of the offset generator keyed by K, and compute K'' from K differently, e.g. by hashing it with a one-way hashing function. To enhance the freshness of K it may incorporate timestamps or nonces[1] typically used in key distribution protocols (e.g. Kerberos [21, 11] and KryptoKnight [7]).

If a candidate K' is not random enough then it should not be used before further processing it. In particular, the bits of K' should not be excessively biased towards 0 or 1 in order to produce, with equal probability, any value for keystream bytes.

If K' is structured or has obvious patterns, we can hash it until we get a new, unstructured value for K'. If the bits of K' are excessively biased towards either 0 or 1, one can also hash them until getting a new value for K', and use it directly (if not excessively biased) or XOR it with the old one in order to reduce the bias. Or one can also pseudo-randomly reverse bits until getting an acceptable balance between zeros and ones. In any case, however, one should guarantee that the algorithm used to compute a good K' takes a limited, and as short as possible, execution time.

The choice of a particular technique for computing good K' values depends mostly on the secret data available to the programs using the generator. For instance, to extend the SSL protocol [9] in order to use our cipher, one could use SSL's *master secrets* to compute K'', or even use them directly, and get K' from the *secure bytes* that SSL computes, using a one-way hashing function, to derive keys and secret MACs (see [9], §8.2.2). This way, K' is secret, not structured, and there is a high probability of getting many different $< K', K'' >$ pairs, even when using the same value for K' (i.e. during the same SSL session). To avoid excessively biased K' values, we can either reverse or not output blocks produced by the hashing function in order to maintain a reduced overall bias of the SSL's secure bytes. In this example the key K, which is only conceptual, includes an SSL master secret and client/server random values (nonces) exchanged in SSL's *hello* messages.

2.3 Rotation and modification of secret frames

The rotation and (possibly sparse) modification of secret frames is performed using pseudo-random offset values produced by the offset generator. This section describes how these values are used.

2.3.1 Rotation

Each L-byte long secret frame is r-byte rotated, where r is a value (modulo L) obtained from the offset generator. The set of rotations used in the generation of each new keystream is a *rotation vector* $\vec{r} = \{r_1, \cdots, r_F\}$, where r_i is used to rotate the i-th secret frame. To minimise the similarity between consecutive keystream frames, it is advised to use F different r values for each

[1] Nonces are, by definition, fresh values.

rotation vector.

There are two basic approaches to handle secret frames' rotations. One is to always use new offsets provided by the offset generator to build rotation vectors, and trust that any particular alignment of secret frames is not repeated before applying a high enough number n of rotation vectors (or, in other words, before randomly modifying a high enough number $n{\cdot}M$ of bytes of each secret frame). This approach strongly depends on the values provided by the offset generator. If n is not usually high enough, there is a risk of producing too close and similar keystream frames, allowing attackers to recover many plaintext bytes by randomly XORing the ciphertext with past cryptanalyzed keystream frames.

The other approach is to build rotation vectors from well chosen values, also derived from the offset generator's pseudo-random output, capable of being used for $L-1$ consecutive rotations without repeating any particular alignment of secret frames. After those $L-1$ rotations, we can either reuse the rotation vector, shuffled or not, or we can repeat the process starting by getting another fresh rotation vector. If we guarantee that all secret frames never get realigned before L consecutive rotations, there is a low probability of getting similar keystream frames when the realignment happens, because in the meanwhile we randomly modified $L{\cdot}M$ bytes of each secret frame.

The second approach guarantees a minimum distance between realignments of secret frames but simplifies brute-force guessing attacks, because it uses less rotation vectors. However, the threat of such attacks can be minimised by correctly choosing the cipher's control parameters (see §3). Consequently, we decided to use the second approach, and we build a new rotation vector after producing $L-1$ keystream frames with the previous one.

To build suitable rotation vectors we use the following heuristic. If L is prime, we build a rotation vector from any F different values provided by the offset generator. If L is not prime, then we use the next three steps to build a rotation vector:

1 - get a value for r_1.

2 - get a different value for r_2 so that:

$r_2 \not\equiv r_1 \pm fac(L) \bmod L$

$|r_1 - r_2|$ is prime relatively to L

where $fac(L)$ is any integer factor of L greater than 1. Empirically, we found out that these rules to filter out r_2 values avoid all realignments of SF^1 and SF^2 before L rotations.

3 - get $F-2$ different values for the remaining elements of the rotation vector.

This heuristic for choosing rotation vectors allows them to be used $L-1$ consecutive times without repeating any realignment of *all* secret frames. The *maximum combinations of rotations* (MCR), i.e. the maximum number of different rotation vectors we can build with the heuristic, is:

$$MCR = \tag{1}$$
$$= \begin{cases} \dfrac{L!}{(L-F)!} & \text{if } L \text{ is prime} \\ \displaystyle\sum_{r_1=0}^{L-1} (L - \mathcal{H}_2(L, r_1)) \cdot \displaystyle\prod_{f=1}^{F-2} (L-f-1) & \text{otherwise} \end{cases}$$

where $\mathcal{H}_2(L, r_1)$ is the number of candidate r_2 values reject by step 2 of the heuristic for a given L and r_1.

2.3.2 Modification

To do M byte modifications in each secret frame we consecutively get M pairs of values (modulo L) from the offset generator – o_{1_j}, o_{2_j}, $j \in \{1, \cdots, M\}$ – and update the o_1-th byte of the i-th secret frame by XORing it with the o_2-th byte of the $i+1$-th secret frame and the o_2-th byte of the plaintext encrypted/decrypted with the previous keystream frame:

$$\begin{aligned} &\forall\, i \in \{1, \cdots, F\}, \\ &\forall\, j \in \{1, \cdots, M\}, \\ &SF^i[o_{1_j}] = SF^i[o_{1_j}] \oplus \\ &\qquad SF^{i+1}[o_{2_j}] \oplus P[o_{2_j}] \end{aligned} \tag{2}$$

were SF^i and P represent the i-th secret frame and the plaintext, respectively, and $SF^{F+1} \equiv SF^1$. Considering all secret frames, the *maximum combinations of modifications* (MCM), i.e. the maximum number of combinations of o_1 and o_2 values that can be used when generating a new keystream frame, is:

$$MCM = L^{2FM} \quad (3)$$

and the maximum combinations of *effective byte modifications* (EBM), assuming $M \leq L$, is:

$$EBM_{max} = \left[\sum_{m=1}^{M} \binom{L}{m} \left(2^8 - 1\right)^m\right]^F \quad (4)$$

where m represents the number of effective byte modifications per secret frame. The value of MCM can be greater or lower than the value of EBM_{max} depending on the values of L, F and M.

By XORing bits from two secret frames we probabilistically reduce the bias of secret frames' contents while modifying them. If $P(0)_{SF}$ and $P(0)_P$ are the probabilities of a bit from secret frames or plaintext being zero, and

$$\begin{aligned} P(0)_{SF} &= 0.5 + \varepsilon^0_{SF} \\ P(0)_P &= 0.5 + \varepsilon^0_P \end{aligned} \quad (5)$$

then, with a modification using the above algorithm, the probability $P(0)_{SF}$ of each new bit is:

$$P(0)_{SF} = 0.5 + 4 \cdot \varepsilon^0_{SF} \cdot \varepsilon^0_{SF} \cdot \varepsilon^0_P \quad (6)$$

which is always closer to 0.5 than the initial one. The value of ε^0_P, which is unpredictable, helps in changing the sign of the expression $\varepsilon^0_{SF} \cdot \varepsilon^0_{SF} \cdot \varepsilon^0_P$:

ε^0_{SF}	ε^0_P	$\varepsilon^0_{SF} \cdot \varepsilon^0_{SF} \cdot \varepsilon^0_P$
> 0	> 0	> 0
> 0	< 0	< 0
< 0	> 0	> 0
< 0	< 0	< 0

Note that an attacker controlling $P(0)_P$ (e.g. using a chosen-plaintext attack) cannot deterministically increase $|\varepsilon^0_{SF}|$; it can only influence the sign of ε^0_{SF}. To reduce that influence one can interleave the usage of either $P[o_2]$ or $\overline{P[o_2]}$, its bitwise inverse, in consecutive modifications. This way, attackers cannot deterministically influence any aspect of the evolution of ε^0_{SF} along the generation of keystream frames (see §4.3).

Using plaintext bytes for modifying secret frames increases the global randomness of the keystream generation algorithm without a significative performance penalty. Furthermore, we introduce an history factor in the generation algorithm, which forces attackers to trace and store many bytes of both plaintext and ciphertext in order to guess the internal state of the generator.

3 Security analysis

The control parameters of our keystream generation algorithm – number of secret frames (F), length of frames (L) and number of byte-modifications per secret frame (M) – influence its memory consumption, performance and security. The memory consumption increases both with L and F, and is not affected by M. Intuitively, the performance of the keystream generator increases with L, and decreases with both F and M; see §4.2 for a more detailed analysis.

In this section we will address the impact of these three parameters in the overall security of the keystream generator. In particular, we will evaluate how they complicate the task of guessing the internal state of the generator, namely the contents of secret frames, when using known-plaintext attacks. The knowledge of the generator's internal state is fundamental for attackers willing to predict the value of future keystream frames, because these are not computed from past keystream bytes but from (modified) past secret frames. Naturally, we assume that for that purpose attackers know the values of the generator's control parameters.

We also assume that attackers use brute force guessing attacks because we know of no simpler attacks. In §5 we highlight some improved attacks that work against similar ciphers and we explain why they do not succeed against our cipher.

3.1 Without modifications – $M = 0$

Assume that an attacker knows two consecutive keystream frames – KF_i and KF_{i+1}. To guess the value of the secret frames that produced KF_i and KF_{i+1} the attacker must guess, for each possible rotation vector $\vec{r_j}$, X bytes of $SF_i^1, \cdots, SF_i^F$ that, together with $\vec{r_j}$, produce $SF_{i+1}^1, \cdots, SF_{i+1}^F$ and, finally, KF_{i+1}.

Empirically, we observe that X depends on L, F, and on the elements of $\vec{r_j}$, being greater or equal than a minimum value X_{min} given by:

$$X(L, F, \vec{r_j}) \geq X_{min} = 1 + L(F-2) \quad (7)$$

Since any value is possible for the contents of the X bytes, we have the following number of possible combinations for guessing correctly $SF_i^1, \cdots, SF_i^F$ and $SF_{i+1}^1, \cdots, SF_{i+1}^F$:

$$C_{i,i+1} = \sum_{j=1}^{MCR} 2^{8\,X(L,F,\vec{r_j})} \quad (8)$$

Now assume the attacker also knows the next keystream frame, KF_{i+2}. From each of the $C_{i,i+1}$ combinations the attacker may now find if a previous rotation vector, or a new one, produce KF_{i+2} from the guessed secret frames. The number of combinations he has to try to guess valid values for secret frames is then, in the worst case:

$$C_{i,i+1,i+2} = MCR \cdot C_{i,i+1} \quad (9)$$

or, in the best case,

$$C_{i,i+1,i+2} = C_{i,i+1} \quad (10)$$

depending if a new rotation vector was fetched or not to generate the $i+2$-th keystream frame.

This result can be generalised to any number of keystream frames, consecutive or not. For non-consecutive keystream frames the attacker can use *equivalent rotation vectors* equal to the sum (modulo L) of all intermediate rotation vectors.

From equation (10) we conclude that the more keystream frames an attacker knows, until a limit of $L-1$, the more he is able to validate correct guesses of secret frames (and rotation vectors). To avoid such danger, there are two non-mutually exclusive solutions: raise $C_{i,i+1}$ to a prohibitive value concerning exhaustive search attacks, or invalidate equations (8) to (10) by using $M \neq 0$, i.e. always performing at least one modification of each secret frame when generating a new keystream frame.

3.2 With modifications – $M \neq 0$

Equations 8 to 10 are valid for $M = 0$, i.e. without considering byte modifications in secret frames. Considering modifications, and equations (3) and (4), equations (8) to (10) must be re-written as follows:

$$Y = min\,(MCM, EBM_{max} + 1)$$

$$C_{i,i+1} = Y \cdot \sum_{j=1}^{MCR} 2^{8\,X(L,F,\vec{r_j})} \quad (11)$$

$$C_{i,i+1,i+2} = Y \cdot MCR \cdot C_{i,i+1} \quad (12)$$

or, in the best case,

$$C_{i,i+1,i+2} = Y \cdot C_{i,i+1} \quad (13)$$

where Y represents the minimum of MCM and $EBM_{max}+1$.

As expected, the number of combinations increases with any of the three parameters, L, F and M. Furthermore, the more keystream frames an attacker has, more combinations he has to try to guess the correct values of secret frames (and rotation vectors). Since the value of F should be as low as possible to increase the performance and reduce the memory consumption of the generator, one should use $F = 2$ or, at the maximum, $F = 3$. For these values of F, and for each value of L, one can derive a minimum value for M in order to get a value for expression (11) higher than a given level. For example, if we have $F = 2$ and $L = 128$, and considering $X(L, F, \vec{r_j})$ always equal to X_{min}, then

$$C_{i,i+1} \geq 2^{32} \implies M \geq 1$$
$$C_{i,i+1} \geq 2^{64} \implies M \geq 2$$
$$C_{i,i+1} \geq 2^{132} \implies M \geq 4$$

Note that the above discussion is only relevant if attackers know the values of L, F and M. If they only know part of them, then the

value of $C_{i,i+1}$ gets higher. As previously stated, these three parameters, or at least one of them, can be chosen dynamically for each case, being thus very easy to complicate exhaustive search attacks to guess the contents of secret frames.

3.3 Influence of the offset generator

The output of EKPRGs used as offset generators has an impact in the overall security of our generator. If the offset generator produces a reduced set of output values, or a well-known sequence of values[2], or even a sequence of values with a known short period, that reduces the values of expressions (1), (3), (4) and (7). This results in a reduction of the maximum value of $C_{i,i+1}$, simplifying the brute-force cryptanalysis of our generator. The exact reduction of $C_{i,i+1}$ can only be computed for specific EKPRGs.

Using cryptographically weak EKPRGs as offset generators does not necessarily reduce the overall security of our generator. For instance, if a given EKPRG fails under ciphertext-only or known-plaintext attacks, like the Gifford's cipher [8], this is not a problem for our generator. In fact, to cryptanalyze the EKPRG, in order to obtain its key K' or its future output, an attacker first has to guess its past output (offset values), which means guessing correctly rotation vectors and modification offsets used by our generator. In other words, it involves an initial and successful cryptanalysis of our generator. As a result, we can use EKPRGs that are not cryptographically strong enough to be used directly as Vernan ciphers without compromising the overall security of our generator.

4 Implementation and evaluation

In this section we describe the implementation of the keystream generator and we evaluate its performance and security.

[2] Like the one produced by the multiplicative congruential pseudo-random number generator implemented by the C library function `rand()`.

4.1 Implementation

Our implementation of the keystream generator supports two values for F (2 and 3) and any value for L up to 256. The generator does not use a specific offset generator; any offset generator capable of producing 1-byte long offsets (modulo L) can be provided at initialisation time. It does not modify source bytes, either plaintext or ciphertext, and was optimised in order to reduce the number of function calls and maximise the use of CPU registers instead of memory accesses. We unrolled encryption/decryption cycles using 8 copies of the loop body, but we did not unroll per frame modification cycles. To speedup the creation of new rotation vectors we use two lists created at initialisation time: one with the factors of L, and the other, for non-prime L values, with values lower than L that are not prime relatively to L. If L is even, however, the heuristic for building rotation vectors has a simple verification that allows us to remove even values from the second list.

To avoid the cost of copying memory buffers when rotating secret frames, they are implemented as two contiguous buffers with equal contents. Rotating secret frames is very efficient, requiring only updating pointers to the actual beginning of secret frames. The drawback of this approach is that we need to duplicate the modifications of secret frames and we spend more memory to store them.

The keystream frame is not actually computed prior to encrypt or decrypt data. Instead, it is used as a repository of ciphertext bytes or plaintext bytes when encrypting or decrypting data, respectively; ciphertext bytes are produced or decrypted using directly the contents of secret frames. When the keystream frame is exhausted, its contents are used to recover plaintext bytes for modifying secret frames, either directly when decrypting data, or by XORing them with the appropriate bytes of secret frames when encrypting data. As a result, decryption is slightly faster than encryption.

4.2 Performance evaluation

In this section we evaluate the performance of our cipher using two different offset generators, which were also used as reference ciphers: 8-bit ARC4 and DES working in 8-bit OFB mode [18]. For these two ciphers we used the code of SSLeay, the public-domain SSL version written by E. Young [9, 23]. The SSLeay's implementation of ARC4 also unrolls encryption cycles using 8 copies of the loop body.

The benchmark applications were executed on a Sun SPARCstation 10/40 with 32-Mbyte RAM running SunOS 4.1.3. Tables 1 and 2 present the average encryption speed of the reference ciphers and our cipher when encrypting 10 Kbyte buffers starting from a warm cache situation (after filling the buffers with data). The values presented were computed from the average elapsed time spent in encrypting every buffer, and average values were computed from 30 runs of benchmark applications.

Stream cipher	Encryption speed (Kbyte / s)
ARC4	2172
DES/OFB	87

Table 1: *Average encryption speed of the reference ciphers – 8-bit ARC4 or DES working in 8-bit OFB – when encrypting of a 10 Kbyte buffer.*

The sixth column of Table 2 presents the encryption speed our cipher when using the offset generator indicated in the first column (ARC4 or DES/OFB) and the control parameters, L, F and M, shown in the second to the fourth columns. The last column presents the speedup of our cipher when compared against the reference cipher, ARC4 or DES/OFB, used as offset generator. Assuming the worst case concerning security, i.e. that the values of control parameters are public, we chose M in order to assure a value for expression (11) higher than 2^{128}. The fifth column of Table 2 presents the logarithm base 2 of expression (11) for the control parameters used in the cipher and using $X(L, F, \vec{r_j}) = X_{min}$.

The values of Table 2 show that the performance of our cipher depends on the performance of the offset generator. Comparing the speed of our cipher against the speed of the ciphers used as offset generators, we observe a small speedup when the offset generator is fast (ARC4), and a very high speedup when the offset generator is slow (DES/OFB). When using ARC4 as reference cipher and offset generator, we achieve a minimum speedup of 7 % and a maximum speedup of 63 %. For DES/OFB we achieve a minimum speedup of 428 % and a maximum speedup of 2556 %.

These performance figures show that, even when comparing against a fast reference cipher, like ARC4, our cipher performs better if the same cipher is used as its offset generator. Naturally, the performance of our cipher could be improved by using faster offset generators, possibly cryptographically weaker than the ones we used.

In general, the performance of the cipher increases with L, mainly because we reduce the number of keystream frames that need to be generated per encrypted buffer. Furthermore, we minimise the dependency of the cipher from the speed of the offset generator because less offsets are needed to rotate and modify secret frames. When $F = 3$ and the offset generator is fast, the value of L has a reduced influence in the speed of the cipher, allowing to choose a value for L taking in main consideration the memory consumption of the cipher.

By increasing F we drastically increase the security of the cipher but we only increase significatively its performance if the offset generator is slow. This different behaviour is explained by two opposite consequences: (i) a performance loss directly caused by the increase of F, and (ii) a performance gain indirectly caused by the reduction of M to the minimum, which can be done without compromising security. By minimising M we reduce the number of random accesses to secret frames to modify them, which in our implementation need to be duplicated, and we also minimise the dependency of the cipher from the speed of the offset generator. This means that with faster offset generators we have

Offset generator	L	F	M	$\log_2(C_{i,i+1})$	Encryption speed (Kbyte / s)	Speedup (%)
ARC4	64	2	5	139	2324	7
		3	1	573	2694	24
	128	2	4	132	3054	41
		3	1	1094	2997	38
	256	2	4	142	3442	58
		3	1	2127	3541	63
DES/OFB	64	2	5	139	459	428
		3	1	573	1091	1154
	128	2	4	132	972	1017
		3	1	1094	1624	1767
	256	2	4	142	1547	1678
		3	1	2127	2311	2556

Table 2: *Average encryption speed of a 10 Kbyte buffer using our cipher and ARC4 or DES/OFB as offset generator. The value of* $\log_2(C_{i,i+1})$ *was computed using* $X(L, F, \vec{r_j}) = X_{min}$. *Speedup values were computed dividing the speed of our cipher by the speed of the reference cipher, ARC4 or DES/OFB, used as offset generator.*

lower performance gains due to the reduction of M, and they can be overcome by the performance loss due to the increase of F.

If all the control parameters of the cipher are secret, or at least L is secret, then we can reduce the values of F and M (if greater than 2 and 1, respectively) and get higher encryption speeds. If we permit secret values for L between L_{min} and L_{max}, we introduce and extra factor of complexity in the cipher's brute-force cryptanalysis equal to $(L_{max} - L_{min})$. This extra factor allows to consider lower values for $C_{i,i+1}$ when computing M (given L_{min} and F) and, consequently, to obtain lower values for M than when L is well-known.

4.3 Security evaluation

Our cipher, unlike most stream ciphers, uses plaintext feedback to modify its internal state. Therefore, it is important to analyse how the plaintext data influences the internal state of the cipher. As stated in §2.3.2, if $P(0)_{SF} = 0.5+\varepsilon^0_{SF}$ is the probability of secret frames' bits being zero, attackers using chosen-plaintext attacks can only influence the sign of ε^0_{SF}, but not its absolute value.

We will now evaluate the evolution of ε^0_{SF} along the encryption of 100 Kbyte plaintext buffers when starting from a non-null value. As plaintext buffers we used two with highly biased data (all zeros and all ones) and another with truly random data (produced by the CryptoLib's true random generator [12]). For the configuration of our the cipher we used $L = 128$, $F = 3$, $M = 1$ and ARC4 as offset generator. The evolution of ε^0_{SF} while encrypting the three plaintext buffers is presented in Figure 3.

As expected, we observe in the figure that, independently from the plaintext, the value of $|\varepsilon^0_{SF}|$ has a tendency to get lower along the generation of keystream frames. Such tendency is more relevant when encrypting random data, and may be slightly reduced when encrypting highly biased data, in particular null data. As suggested in §2.3.2, a simple solution to avoid the distinguished influence of opposite highly biased plaintext data (p) is interleaving the usage of either p and $\overline{p}$, its bitwise inverse, on each modification of secret frames. Figure 4 shows that this simple modification, which involves a negligible performance penalty, drastically reduces the effect of plaintext bytes in the evolution of ε^0_{SF} when using the previous evaluation scenario.

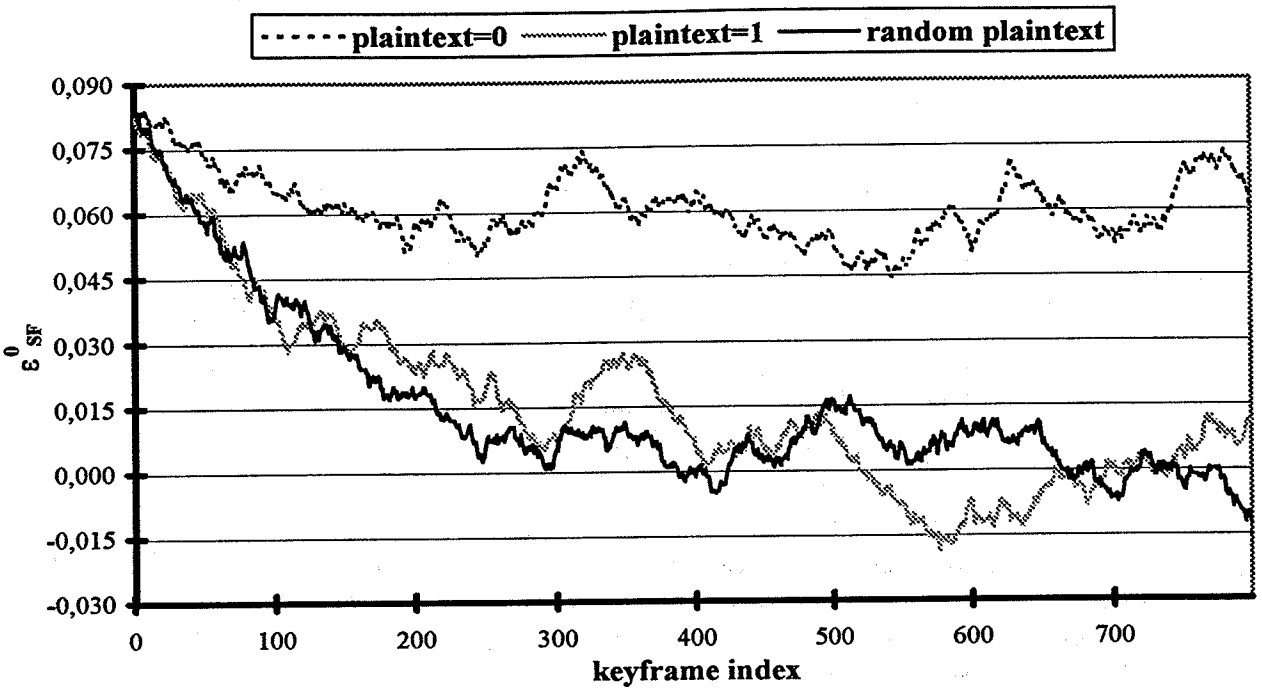

Figure 3: *Evolution of ε^0_{SF} when using two highly biased plaintext buffers (all zeros and all ones) and a plaintext buffer with random data.*

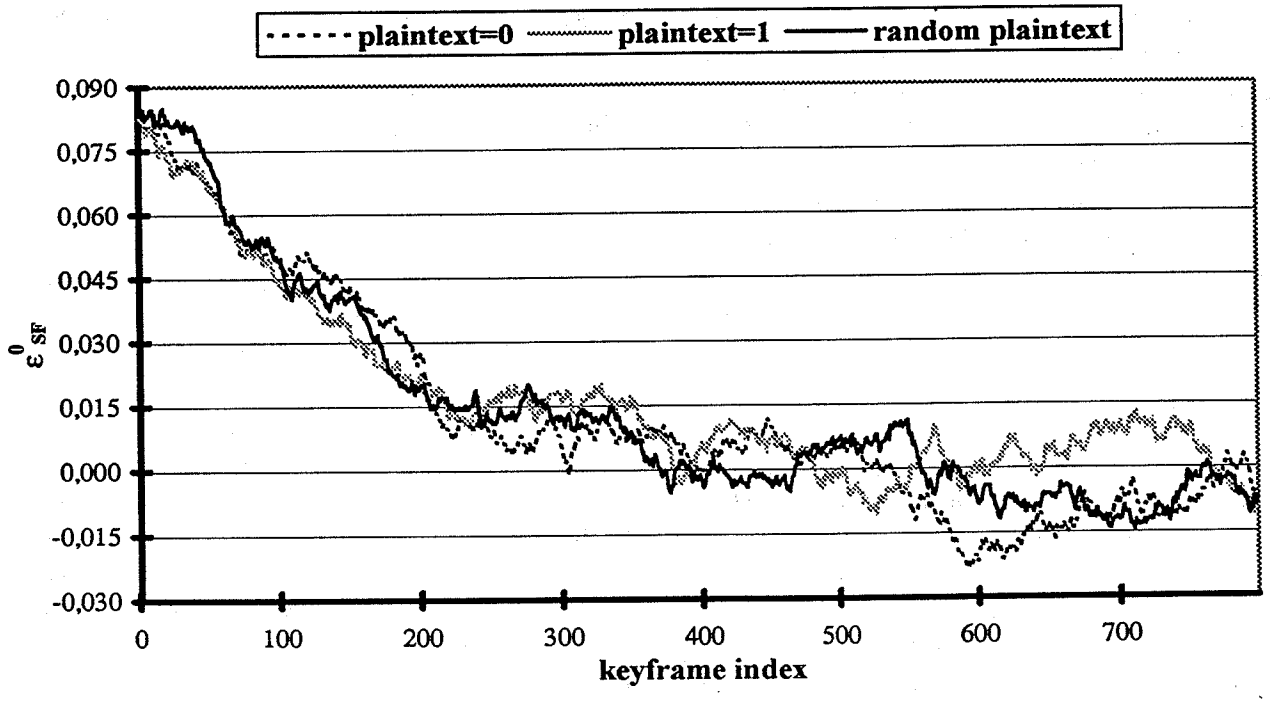

Figure 4: *Evolution of ε^0_{SF} when using two highly biased plaintext buffers (all zeros and all ones) and a plaintext buffer with random data. In this case we interleaved the usage of plaintext bytes and their bitwise inverse while modifying secret frames.*

5 Related work

The design of this keystream generator was inspired in several generators producing bit sequences by XORing the output of several linear feedback shift registers (LFSRs), like the Alternating Step Generator and many others [2, 10, 25, 22, 18]. Considering that the contents of secret frames are the output of LFSRs (*equivalent LFSRs*), and without considering their modification ($M = 0$), our generator is comparable to an extension of the Alternating Step Generator using F LFSRs instead of two to compute output bits, and clocked by the sum of a fixed frequency and pseudo-random high-frequency bursts (the rotation of secret frames). The contents of secret frames would be equivalent to cycles of the equivalent LFSRs.

By considering modifications, the contents of secret frames become equivalent to partial views of possible large and noncyclic outputs of the equivalent LFSRs. However, and unlike for LFSRs, the modification of secret frames is partially independent of their own contents because it is controlled by offsets provided by the offset generator and by plaintext data.

If the output of the offset generator is unknown to attackers, then the linear complexity and the maximal sequence length of the equivalent LFSRs are unpredictable, and the same is true for the output of the generator. As a result, the generator becomes immune to the Siegenthaler's correlation attack, because this attack is based on the statistical analysis of maximal length sequences produced by internal LFSRs with known length [20]. Similarly, it also becomes immune to the linear syndrome attack, because this attack relies on the knowledge of feedback polynomials of the equivalent LFSRs [24].

6 Conclusions

In this paper we presented a new approach in the design and implementation of stream ciphers. The approach consists in allowing ciphers to have a variable internal state with a structure and an evolution model chosen at instantiation time. Arbitrarily large internal states can be used in order to defeat brute-force guessing attacks without compromising the performance of cipher, and even improving it in some cases. The cipher uses an external keyed pseudo-random generator, EKPRG, either cryptographically strong or weak, and plaintext feedback as sources of data for pseudo-randomly modifying its internal state.

The structure and behaviour of the cipher is ruled by three control parameters – the number (F) and length (L) of secret frames (L) and the number of modifications per secret frame (M). These parameters have a different impact in several aspects of the cipher, like its strength against exhaustive search attacks, its memory consumption, and its performance. In particular, the strength of the cipher increases with all of them, the memory consumption increases with F and L, and the performance increases with L and decreases with the other two.

In the security analysis of the cipher we have evaluated how the strength of the cipher against known-plaintext attacks depends on the control parameters, and how these can be chosen in order to achieve arbitrary security levels. From that evaluation we chose several sets of control parameters assuring high security levels and we measured the performance of the cipher when using them. Since the performance of the cipher also depends on the speed of the EKPRG, we used two different and cryptographically strong EKPRGs, one fast (8-bit ARC4) and one slow (DES working in 8-bit OFB). To get a notion of the cipher's speed, we compared it against the speed of two reference ciphers, the same we used as EKPRGs of our cipher.

We evaluated the encryption speed of our cipher, and the two reference ciphers, when encrypting 10 Kbyte buffers starting from a warm cache situation. When comparing the speed of our cipher against the speed of the reference ciphers used as EKPRGs, we obtained a minimum encryption speedup of 7 % and 428 %, respectively, and a maximum speedup of 63 % and 2556 %. These performance results could be improved by using faster EKPRGs, possibly cryp-

tographically weaker than the ones we used.

Since our cipher uses pseudo-random plaintext samples to randomise its internal state, we have experimentally evaluated how the plaintext influences the bias of the secret frames' contents. For the evaluation we used two plaintext buffers with highly biased data and a buffer with truly random data. From the evaluation we confirmed that, as expected, the bias of the secret frames' contents as a tendency to get lower as more keystream frames are generated. In addition, we saw that by interleaving the usage of plaintext bytes and their bitwise inverse, we prevent chosen-plaintext attacks from deterministically influencing the evolution of the bias.

References

[1] Henry Beker and Fred Piper. *Cipher Systems: The Protection of Communications.* Northwood Books, London, 1982.

[2] T. Beth and F. C. Piper. The Stop-and-Go Generator. In *Advances in Cryptology - EUROCRYPT '84 Proceedings*, pages 88–92. Springer-Verlag, 1984.

[3] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In *Advances in Cryptology - EUROCRYPT '93 Proceedings*, pages 398–409. Springer-Verlag, 1994.

[4] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO '90 Proceedings*, pages 2–21. Springer-Verlag, 1991.

[5] E. Biham and A. Shamir. Differential Cryptanalysis of Feal and N-Hash. In *Advances in Cryptology - EUROCRYPT '91 Proceedings*, pages 1–16. Springer-Verlag, 1991.

[6] E. Biham and A. Shamir. Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer. In *Advances in Cryptology - CRYPTO '91 Proceedings*, pages 156–171. Springer-Verlag, 1992.

[7] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution. *ACM Trans. on Networking*, pages 31–41, February 1995.

[8] Thomas R. Cain and Alan T. Sherman. How to Break Gifford's Cipher. In *Proc. of the 2nd Annual ACM Conf. on Computer and Comm. Security*, pages 198–209. ACM Press, 1994.

[9] Alan O. Freier, Philip Karlton, and Paul C. Kocher. SSL Protocol Version 3.0. Internet Draft, Netscape Communications Corp., March 1996.

[10] C. G. Günther. Alternating Step Generators Controlled by De Bruijin Sequences. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology - EUROCRYPT '87 Proceedings, Lecture Notes in Computer Science*, volume 304, pages 5–14. Springer-Verlag, Berlin, 1988.

[11] John T. Kohl. The Evolution of the *kerberos* Authentication Service. In *Spring 1991 EurOpen Conference*, Tromsø, Norway, 1991.

[12] J. B. Lacy, D. P. Mitchell, and W. M. Schell. CryptoLib: Cryptography in Software. In *Proc. of the 4th UNIX Security Symposium*, pages 1–17. USENIX Association, 1993.

[13] X. Lai, J. Massey, and S. Murphy. Markov Ciphers and Differencial Cryptanalysis. In *Advances in Cryptology - EUROCRYPT '91 Proceedings*, pages 17–38. Springer-Verlag, 1991.

[14] James L. Massey. Shift-Register Systhesis and BCH Decoding. *IEEE Trans. on Information Theory*, IT-15(1):122–127, January 1969.

[15] M. Matsui. Linear Cryptanalisys Method for DES Cipher. In *Advances in Cryptol-*

ogy – EUROCRYPT '93 Proceedings, pages 386–397. Springer-Verlag, 1994.

[16] The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms. RFC 2040, RSA Data Security, Inc., MIT Laboratory for Computer Science, October 1996.

[17] R. Rivest. The RC5 Encryption Algorithm. In *2nd Int. Workshop on Fast Software Encryption, Lecture Notes in Computer Science*, pages 86–96, Leuven, Belgium, 1995. Springer-Verlag.

[18] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Wiley & Sons, Inc., second edition, 1996.

[19] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. on Information Theory*, IT-31(5):776–780, September 1984.

[20] T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Trans. on Computers*, C-34(1):81–85, January 1985.

[21] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proc. of the USENIX Winter Conf.*, pages 191–202, Dallas, Texas, USA, February 1988.

[22] S. B. Xu, D. K. He, and X. M. Wang. An Implementation of the GSM General Data Encryption Algorithm A5. In *CHINACRYPT '94 Proceedings*, pages 287–291, Xidian, China, November 1994.

[23] E. A. Young. SSLeay and SSLapps FAQ, 1997. `http://www.psy.uq.edu.au:8080/~ftp/Crypto`.

[24] K. C. Zeng and Huang M. Q. On the Linear Syndrome Method in Cryptanalysis. In *Advances in Cryptology – CRYPTO '88 Proceedings, Lecture Notes in Computer Science*, volume 403, pages 469–478. Springer-Verlag, New York, 1988.

[25] K. C. Zeng, C.-H. Yang, and T. R. N. Rao. On the Linear Consistency Test (LCT) in Cryptanalysis. In *Advances in Cryptology – CRYPTO '89 Proceedings*, pages 164–174. Springer-Verlag, 1990.

A Systematic Procedure for Applying Fast Correlation Attacks to Combiners with Memory

M. Salmasizadeh[1,2], J. Golić[3], E. Dawson[1], and L. Simpson[1]
[1] Information Security Research Centre, Queensland University of Technology
GPO Box 2434, Brisbane Q 4001, Australia
Email: { salmasi, dawson, simpson}@fit.qut.edu.au
[2] Electronic Research Centre, Sharif University of Technology
P.O. Box 11365-8639, Tehran, Iran
[3] Faculty of Electrical Engineering, University of Belgrade
Bulevar Revolucije 73, 11001 Belgrade, Yugoslavia
Email: golic@galeb.etf.bg.ac.yu

Abstract

A systematic procedure for applying fast correlation attacks to combiners with memory is introduced. This procedure consists of the following four stages: identifying correlated linear input and output transforms with maximum possible or relatively large correlation coefficient, calculating low-weight polynomial multiples based on the identified input linear transform, applying an iterative error correction algorithm to the linear transform of the observed keystream and solving several sets of linear equations to determine the initial state of the input LFSRs. This procedure is successfully applied to three keystream generators, namely, the summation generators with three and five inputs, the nonlinear filter generator and the multiplexed sequence generator.

1 Introduction

A well-known type of keystream generator for stream cipher applications consists of a number of linear feedback shift registers (LFSRs) combined by a memoryless nonlinear function. The keystream sequences produced by these generators can achieve desirable cryptographic properties such as a large period, high linear complexity and statistics which satisfy standard statistical tests [20]. However, in [22] and [23] these structures are shown to be vulnerable to divide and conquer correlation attacks based on the termwise correlation between the keystream sequence and a subset of the underlying LFSR sequences. In [22], the concept of the correlation immunity of boolean functions is introduced, and the trade-off between linear complexity and correlation immunity pointed out. According to [25], the output of any memoryless boolean function is correlated to at least one linear function of its inputs. Once such a linear function has been determined, it can then be used to mount a basic correlation attack on the involved inputs. More importantly, fast correlation attack techniques based on iterative probabilistic decoding have been used and introduced in [17], see also [4], [18], [19] and [10]. These attacks are successful if the correlation coefficient is large enough, and if the LFSR feedback polynomials involved have sufficiently many low-weight (weight is the number of nonzero terms) polynomial multiples of moderately large degrees, see the

convergence condition [19].

To overcome the trade-off between linear complexity and correlation immunity, Rueppel [20] suggested the use of combiners with memory. The combiner with memory, as shown in the inset on Figure 1, is a non-autonomous finite-state machine [8], defined by

$$S_{t+1} = F(X_t, S_t), \quad t \geq 0 \tag{1}$$

$$y_t = f(X_t, S_t), \quad t \geq 0 \tag{2}$$

where, at time t, $X_t = (x_{1t}, \cdots, x_{nt})$, is an input vector, $S_t = (s_{1t}, \cdots, s_{mt})$, is a state vector, with S_0 the initial state, and y_t is the output bit. The function $F : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^m \rightarrow \mathrm{GF}(2)^m$ is a next state vector Boolean function, and $f : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^m \rightarrow \mathrm{GF}(2)$ is an output Boolean function. The notations $F(X, S)$ and $f(X, S)$ are used for the next state and output functions, respectively.

Correlation properties of a general combiner with an arbitrary number, m, of memory bits are analysed in [6]. It is shown that in such a combiner there exists a nonzero linear function (transform) of at most $m+1$ successive output bits that is correlated to a linear function of at most $m+1$ successive input binary vectors. For a combiner with m bits of memory and n inputs, the required computational complexity to determine the correlation coefficients between a given linear function of $m+1$ successive outputs and all linear functions of $m+1$ successive inputs, using the Walsh transform technique, is $O((mn + m + n)2^{mn+m+n})$. Considering all $2^{m+1} - 1$ nontrivial output linear functions, the total computational complexity becomes $O((mn + m + n)2^{mn+2m+n})$. This is not feasible for relatively large mn which, according to [8], is needed to obtain a sufficiently small value of the largest correlation coefficient. The linear sequential circuit approximation (LSCA) method as introduced in [6] provides a feasible procedure for finding such pairs of linear functions with comparatively large correlation coefficients. The LSCA method consists of determining and solving a linear sequential circuit that approximates a binary combiner with memory. This method generally applies to arbitrary binary combiners with memory without any restriction regarding the output and next-state functions.

The first objective of this paper is to introduce a systematic procedure for cryptanalysis of regularly clocked linear feedback shift register (LFSR) based stream ciphers whose keystream generators are combiners with memory. While cryptanalysis of particular keystream generators based on the correlation weakness has been published, a comprehensive general method as presented in this paper has not been published as such. It is assumed that the cryptanalyst knows the complete structure of the keystream generator, including the number of input LFSRs and memory bits, the feedback polynomials of the LFSRs, the output and the next-state vector Boolean functions. In addition, it is assumed that a segment of the keystream sequence is known. The goal is to determine the secret key, which controls the initial state of the input LFSRs. The second objective is to demonstrate the applicability of this method of cryptanalysis to three well known keystream generators: the summation generator [21], the nonlinear filter generator [9] and the multiplexed sequence generator [12].

2 Fast Correlation Attacks

2.1 Probabilistic model

The fast correlation attack is based on a model in which the observed keystream sequence $z = \{z_i\}_{i=1}^N$ is regarded as a noisy version of an underlying LFSR sequence $a = \{a_i\}_{i=1}^N$. That is, the sequence $z = \{z_i\}_{i=1}^N$ is the output of a memoryless binary symmetric channel (BSC) with error probability p (corresponding to c, the known correlation coefficient, where $c = 1 - 2p$) when the unknown LFSR sequence $a = \{a_i\}_{i=1}^N$ (to be reconstructed) is the input. An iterative, probabilistic, parity-check based algorithm is used to perform error correction in an attempt to reconstruct a.

2.2 Parity-checks

A parity-check is any linear relationship satisfied by a LFSR sequence. It is known that the parity-checks correspond to polynomial multiples $h(x)$, of the LFSR feedback polynomial $f(x)$ such that $h(0) = 1$ [4]. For the fast correlation attack, the objective is to obtain sufficiently many parity-check polynomials $h(x)$ of low weight and of as small degree as possible, because the maximum degree used determines the length of keystream sequence required for the attack.

A number of methods can be used to generate low weight polynomial multiples. If $f(x)$ is of low weight, repeated squaring [17] of $f(x)$ is a simple weight-preserving technique. If not, other methods must be used. To generate all $h(x)$ of weight at most 5 and degree at most $M \geq r$, where r is the degree of $f(x)$, we used the polynomial residue method based on the birthday paradox, briefly mentioned in [17]. First, in $O(M)$ time, compute and store the residues of all the monomials $x^m \mod f(x)$, $1 \leq m \leq M$. Second, in $O(M^2/2)$ time, compute (by bitwise summation) the sums $x^i + x^j \mod f(x)$ and store the residues, for all different pairs $1 \leq i < j \leq M$. Third, in $O(M^2 \log_2 M)$ time, use a fast sorting algorithm to sort these summation residues. A pair of sums which have equal residues can be used to form a polynomial multiple of weight 4 (weight 2 is not possible if M is smaller than the period of $f(x)$). A pair of sums for with residues which differ by 1 (the binary sum of the residues is equal to 1) can be used to form a polynomial multiple of weight 3 or 5. By the birthday paradox argument, expected to be valid for a random $f(x)$, equal residues will, with high probability, appear if $M^2 \geq 2^{r/2}$, i.e., if $M \geq 2^{r/4}$, because the equivalent number of residues tested to find equal residues is $2\binom{M}{2}$. In fact, pairs of sums with residues which differ by 1, yielding polynomial multiples of weight 5, are much more likely than the others, a the point overlooked in [17] where such pairs are not considered at all. So, both the required precomputation time and the storage required for finding the parity-check polynomials of weight at most 5 are $O(2^{r/2})$. A similar method can be used to generate all $h(x)$ of weight at most $2k + 1$, but this requires the computation and storage of the residues $x^{i_1} + \cdots + x^{i_k} \mod f(x)$ for all $\binom{M}{k}$ combinations $1 \leq i_1 < \cdots < i_k \leq M$, see [7].

2.3 Iterative error-correction algorithm

In this paper an iterative probabilistic, parity-check based decoding algorithm [18], with a modification given in [10], is applied. The algorithm consists of a number of rounds, each composed of several iterations. First, for each of N observed keystream bits, a set of preferably orthogonal parity-checks is first determined. The algorithm starts with the observed keystream sequence $\{z_i\}_{i=1}^{N}$ and with $p < 0.5$ as the prior probability of error for each bit. The observed keystream sequence is then iteratively modified to yield the reconstructed LFSR sequence. Each iteration consists of the following major steps:

- *Step 1:* For each $z_i, i = 1, \ldots, N$, calculate the parity-check values.
- *Step2:* Use the parity check values for each z_i to compute the posterior probability of error, p_i, using the posterior probabilities of error from the previous iteration as the prior probabilities in the current iteration, see [10].
- *Step 3 (error-correction):* If $p_i > 0.5$, then set $z_i = z_i \oplus 1$, and $p_i = 1 - p_i$, $i = 1, \ldots, N$

If p is not too close to 0.5, then most of the error probabilities typically quickly converge to zero, and the number of errors is reduced, but not necessarily to zero. Further errors can be corrected by resetting the error probabilities to p and repeating the algorithm for several rounds. Finally, a simple (sliding window) information set decoding technique is used to recover the LFSR initial state. In fact, we apply an improved algorithm with the so-called fast resetting and with the sliding window technique incorporated in the rounds, see [10].

3 Systematic Procedure for Fast Correlation Attacks on Combiners with Memory

This section of the paper outlines the procedure for attacking combiners with memory. The procedure consists of four main stages. Firstly, of all the nonzero linear functions of at most $m+1$ successive output bits, and all the linear functions of at most $m+1$ successive input binary vectors, a pair of input and output linear functions with the maximum possible or a comparatively large correlation coefficient is determined. (NOTE: there may be more than one pair of functions with the maximum correlation coefficient.) This can be achieved by exhaustive search (if feasible), by the LSCA method or by analytical methods. In the second stage, parity-check polynomials are generated. Sufficiently many low-weight polynomial multiples of the least common multiple of the LFSR feedback polynomials involved in the input linear function (identified in first stage) have to be generated. A polynomial residue method based on the birthday paradox is used for this. The third stage is the application of the iterative error correction algorithm. The inputs required by the algorithm are: an observed keystream sequence or a linear transform of it, the correlation coefficient (determined in the first stage) and the set of parity-check polynomials (determined in the second stage). If the attack is successful, the output of the algorithm is a linear transform of input sequences, corresponding to one of the input linear functions identified in the first stage. The final

stage of the procedure involves a search for the correct input linear function, and solving a set of linear equations to determine the secret key. The four stages, and the relationships between them, are shown in Figure 1.

4 Cryptanalysis of the Summation Generator with Three and Five Inputs

4.1 Description of the summation generator

The *summation generator* [20], [16] is a binary nonlinear combiner with memory whose internal state variable, the carry, takes integer values from the set $[0, n-1]$, where n is the number of inputs. An initial value for the carry is assumed: zero, or any value from $[0, n-1]$. The memory size in bits is thus $m = \lceil \log_2 n \rceil$. Let $X_t = (x_{1,t}, \ldots, x_{n,t})$ and y_t denote the n input bits and one output bit at time t, and let S_t and $S_t^{(0)}$ denote the carry and the least significant bit of the carry at time t, respectively. Then the output and the next-state functions of the summation generator are, for $t \geq 0$, defined by

$$y_t = \bigoplus_{i=1}^{n} x_{i,t} \oplus S_t^{(0)} \tag{3}$$

$$S_{t+1} = \left\lfloor \left(\sum_{i=1}^{n} x_{i,t} + S_t \right) /2 \right\rfloor \tag{4}$$

with modulo 2 summation in (3) and integer summation in (4). The input sequences are defined as the LFSR sequences generated from distinct primitive feedback polynomials. The LFSR initial states are controlled by the secret key, and the initial carry, S_0, is either fixed or also controlled by the secret key. Due to the binary summation in (3), the summation generator is maximum-order correlation immune [20], that is, for any given initial carry, the output sequence is (statistically) independent of any proper subset of the input sequences, where these are assumed to be purely random. As a consequence, any linear transform of the input sequences that is correlated to a linear transform of the output sequence must involve all input sequences.

4.2 Theoretical analysis

The essence of the LSCA method [8] is in finding good linear approximations to the output and component next-state functions, and solving the resulting linear sequential circuit. In the case of the summation generator, the output function is already linear, and the least significant carry bit can be approximated as 1 with correlation coefficients $1/3$ and $-2/15$ for three and five input summation generators, respectively, according to the asymptotic probability distribution derived in [24]. These correlation coefficients are large enough to apply the fast correlation attack. In fact, for the three input summation generator, the correlation coefficients between all linear functions of three $(m+1)$ successive outputs and inputs,

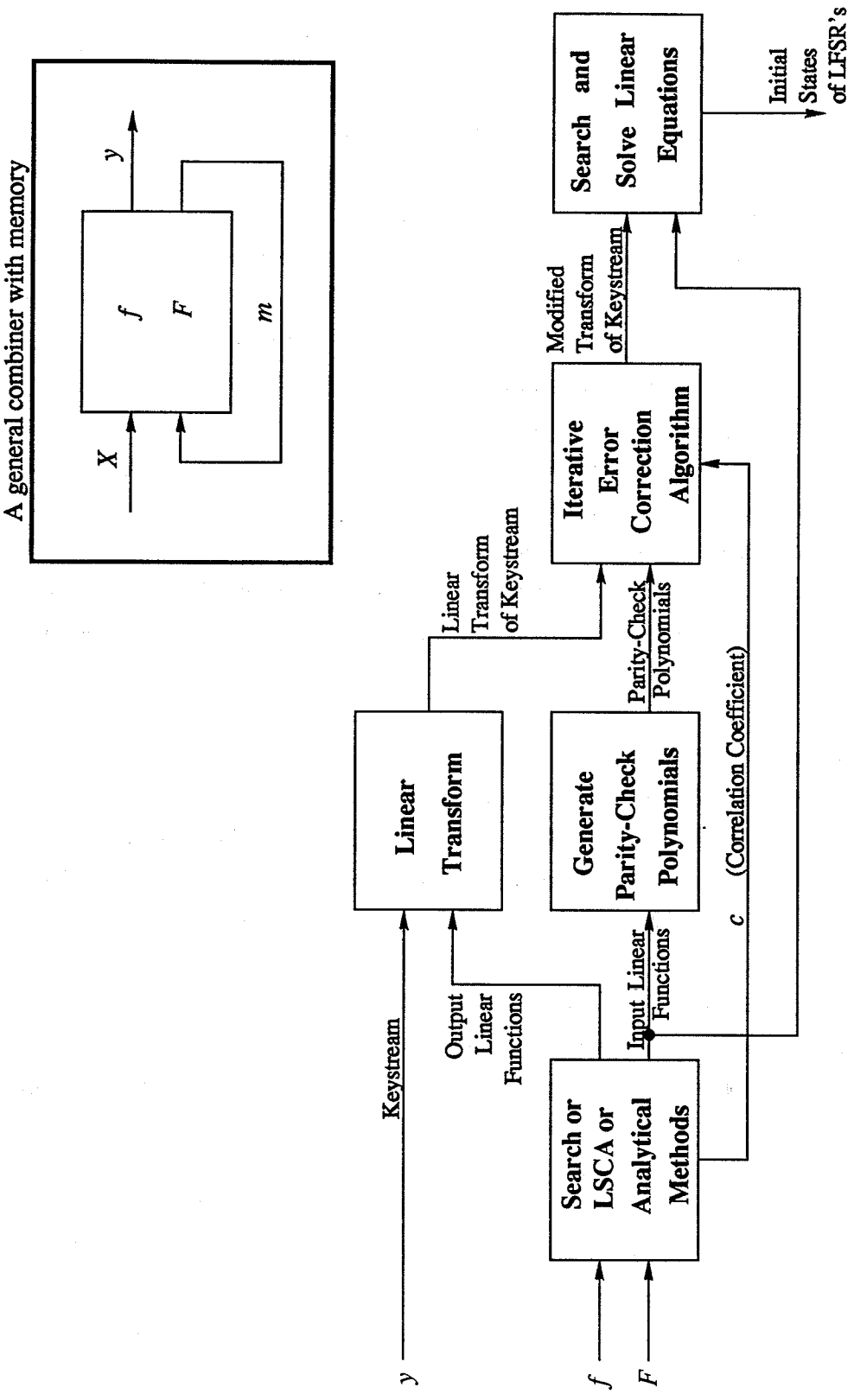

Figure 1: A systematic procedure for applying fast correlation attacks to combiners with memory.

were computed, and the largest absolute value obtained was 1/3, well distinguished from the next closest coefficients. In addition, the output bit complemented is correlated to only one input linear function (the binary sum of the current input bits) with correlation coefficient 1/3. Accordingly, the iterative error-correction algorithm as described in Subsection 2.3 should, if successful, modify the keystream sequence to converge to this linear transform. In the case of summation generator with five inputs, there exist six and sixteen input linear functions correlated to the output bit with correlation coefficients equal or close to 2/15 and to $-2/15$, respectively. The feedback polynomial of the resulting LFSR sequence to be reconstructed, see Subsection 2.1, is the product of the feedback polynomials of the input LFSRs, assumed to be primitive, distinct and known to the cryptanalyst. The parity-checks to be used in the attack can be obtained by the method described in Subsection 2.2. According to the convergence condition [19], the minimum number of of parity-checks of weight w required per keystream bit depends only on the correlation coefficient and w. However, in our case more are required because the parity-checks are not quite orthogonal, the output sequence length is minimised (and hence close to the maximum degree of the parity-check polynomials used) and the errors are not memoryless as in the BSC model. For large r and a fixed parity-check weight w, both the keystream sequence length required and the computational complexity are $O(2^{r/(w-1)})$, and for each r there is an optimal weight w that minimises the computational complexity.

4.3 Experimental results

For the three input summation generator experiments were conducted on four different generators (different LFSR lengths and tap settings). Similarly, for the five input summation generator, two different generators were used. For each generator, twenty LFSR initial states were generated randomly; the keystream produced and the attack conducted for each state. For the three and five input summation generators, experiments were conducted with parity-checks predominantly of weight six and seven, or five, respectively (depending on r). For each of the 20 initial states, both the number of parity-checks and the keystream length used were minimised. For successful experiments, the degree r; the weight w_p of the product feedback polynomial; the weight w, average number K_{av}, maximum number K_{max} and maximum degree M_{max} of the parity-check polynomials used, and also the keystream sequence length N are all shown in Tables 1 and 2. The parity-checks used were not necessarily orthogonal.

For comparison, the keystream length required and the computational complexity of the divide-and-conquer attack [5], the 2-adic complexity attack [15] and the fast correlation attack on both three and five input summation generators (assuming for simplicity that the LFSR lengths are equal), are shown in tables 3 and 4 respectively. These tables demonstrate the improvement provided by the fast correlation attack.

Table 1: The three input summation generator - parity-check polynomial information and keystream sequence length.

r	w_p	w	K_{av}	K_{max}	M_{max}	N
30	17	5	4	6	559	$2^{9.52}$
		6	425	518	585	
		7	282	282	250	
34	19	5	13	152	800	$2^{9.95}$
		6	146	240	849	
		7	5360	5945	650	
38	19	6	19	19	830	$2^{10.67}$
		7	5360	5945	650	
44	17	7	2600	2600	1800	$2^{11.37}$

Table 2: The five input summation generator - parity-check polynomial information and keystream sequence length.

r	w_p	w	K_{av}	K_{max}	M_{max}	N
30	13	5	8338	9272	3915	$2^{12.1}$
		6	598	598	604	
		7	130	130	223	
40	21	5	14800	14800	24968	$2^{14.9}$

Table 3: Comparing complexity of attacks, 3 input summation generator.

	Div. and Conq.	2-adic	Fast Corr.
Required Length	$O(r)$	$O(2^{r/3})$	$O(2^{r/4})$
Complexity of Attack	$O(2^{2r/3})$	$O(2^{2r/3})$	$O(2^{r/4})$

Table 4: Comparing complexity of attacks, 5 input summation generator.

	Div. and Conq.	2-adic	Fast Corr.
Required Length	$O(r)$	$O(2^{r/5})$	$O(2^{r/4})$
Complexity of Attack	$O(2^{4r/5})$	$O(2^{2r/5})$	$O(2^{r/4})$

5 Cryptanalysis of the Nonlinear Filter Generator

5.1 Description of the Nonlinear Filter Generator

The nonlinear filter generator (NLFG) consists of a single regularly clocked binary linear feedback shift register (LFSR) of length r and a nonlinear Boolean function f of n input variables. The keystream is generated by applying f to the output of n stages of the LFSR. Let the LFSR output sequence be denoted as $a = \{a_i\}$ and the sequence of inputs to the filter function as $X = \{X_i\}$, $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\}$ defined as $X_i = \{a_{i+\gamma_1}, \ldots, a_{i+\gamma_n}\}$, where the tapping sequence $\gamma = (\gamma_1)_{i=1}^{n}$ is an increasing sequence of nonnegative integers such that $\gamma_n \leq r - 1$. The generator output sequence is denoted $z = \{z_i\}$, with $z_i = f(X_i)$. For cryptographic properties of NLFGs, see [20]. Also, the NLFG can be viewed as a finite input memory combiner with one input and one output, with memory size given by $M = \gamma_n - \gamma_1$ [9].

5.2 Theoretical analysis

In this section, the security of the NLFG with respect to the procedure proposed in Section 3 is investigated. The observed segment of the keystream sequence $z = \{z_i\}_{i=1}^{N}$ is regarded as a noisy version of a segment of a nontrivial linear transform of the unknown LFSR sequence a, that is, $z_i = l(X_i) + e_i$, where $l(X_i) = \sum_{j=1}^{n} c_j x_{i,j}$, $c_j \in \{0, 1\}$, not all c_j are equal to zero, the summation is modulo two and $\{e_i\}_{i=1}^{N}$ is a segment of a binary noise sequence with $\Pr(e_i = 1) = p < 0.5$ for each $i = 1, 2, \ldots, N$. The corresponding correlation coefficient is defined as $c = 1 - 2p > 0$ and is equal to the correlation coefficient between f and l, that is, $p = \Pr(f \neq l)$. Note that any (feedforward) linear transform of a is also a linear recurring sequence satisfying the same feedback polynomial as a.

The linear function l specifying a linear transform of a is to be determined by the attack. The correlation coefficients between f and all the linear functions of the same n variables are precomputed by the Walsh transform technique [20] with computational complexity $O(n2^n)$. Let $\mathcal{L}^+$ and $\mathcal{L}^-$ denote the sets of linear functions with positive and negative correlation coefficients, respectively. Since the success of the attack predominantly depends on how large c is, if the maximum absolute value of c is associated with the set $\mathcal{L}^-$, the complement of the keystream is used in the attack. For a fixed value of n, the value of p varies depending on the particular Boolean function used as a filter. In general, for random balanced functions, the expected value of p increases as n increases.

The set of parity-checks used is formed from phase shifts of the parity-check polynomials, these are obtained by repeated squaring of the LFSR feedback polynomial, assumed to have a low weight. If the LFSR polynomial does not have a low weight (e.g., if its weight is 10 or bigger), then the method described in Subsection 2.2 can be applied to obtain low-weight polynomial multiples [7]. In any case, if the weight of the parity-checks used is not low, then more such parity-checks are required and the complexity of the fast correlation attack may become prohibitively high. The iterative probabilistic error-correction algorithm described in Subsection 2.3 is used to make modifications to the keystream sequence to reconstruct a linear transform of the LFSR sequence a.

The final stage of the systematic procedure involves recovering the unknown LFSR initial state. The reconstructed LFSR sequence can be obtained as a linear transform, $l \in \mathcal{L}^+$, or $l \in \mathcal{L}^-$ of the original LFSR sequence a, depending on whether the keystream or its complement is used in the attack. Thus all such candidate linear transforms should be tested, in order of decreasing correlation coefficients. Consequently, for any assumed l, the candidate LFSR initial state is obtained from any r consecutive bits of the reconstructed LFSR sequence by solving the corresponding nonsingular system of linear equations. Each candidate initial state is then used to generate a candidate keystream by the known filter function which is compared with the original keystream. If they are identical, then the correct LFSR initial state is found and is very likely to be unique.

5.3 Experimental results - random balanced filter functions

In these experiments, we used a 63-bit LFSR with a primitive feedback polynomial, $1+x+x^{63}$, of weight three and a set of parity-checks formed from the phase shifts of the parity-check polynomials obtained by repeated squaring of the LFSR feedback polynomial. The attack works similarly on larger LFSR lengths. The experiment was conducted for $n = 5, 6, 7, 8$, and 9. For each n, the experiments were performed for both consecutive (CONS) and full positive difference set (FPDS) tapping sequences. Note that FPDS tapping sequences are much more resistant to the conditional correlation attack [3] and the inversion attack [9]. Also, two different parity-check (polynomial) sets were used: a set of 6 parity-checks and the observed keystream length of 2500 bits and a set of 9 parity-checks and the observed keystream length of 20000 bits. For each of these four sets of conditions (tapping sequence and number of parity-checks) and each n, ten different *balanced* filter functions and twenty different nonzero LFSR initial states were randomly chosen.

Our main observation is that the iterative error-correction algorithm, if successful, mainly converges to one of the best linear transforms of the original LFSR sequence, or to a linear transform that is close to being the best with respect to the correlation coefficient absolute value. Figure 2 shows the success rate of the performed attacks for various values of the theoretical probability of noise, p, under each set of conditions. The value of p was obtained by rounding the probability of noise for randomly chosen filter functions. As expected, the graph shows that the proportion of successful attacks generally decreases as the probability of noise, p, increases; that the success of the attack is practically independent of the tapping sequence; and that for a given p, increasing the number of parity-checks and consequently the observed keystream length increases the success rate. Irregularities in the graph are due to the small number of filter functions for particular values of p.

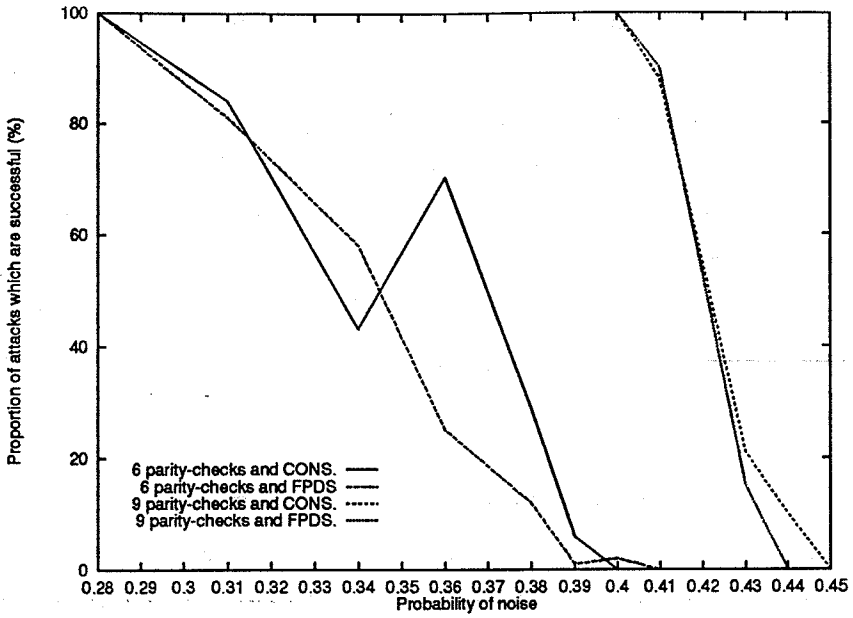

Figure 2: Success rate versus theoretical noise probability p.

6 Cryptanalysis of the Multiplexed Sequence Generator

6.1 Description of the Multiplexed Sequence Generator

A class of binary pseudorandom sequences for cryptographic and spread spectrum applications called the multiplexed sequences was proposed and analysed in [12, 13, 14]. Their use has also been recommended in an EBU standard for video encryption for pay-TV [1]. Multiplexed sequences are generated by a simple and fast scheme consisting of two linear feedback shift registers and a multiplexer whose address is controlled by one of the shift registers and whose inputs are taken from the other. They were shown to possess good standard cryptographic properties such as long period, high linear complexity and low out-of-phase autocorrelation.

Multiplexed sequences are defined here as a slight generalisation of the class of binary sequences introduced and analysed in [12]. Let $a = (a(t))_{t=0}^{\infty}$ be a binary maximum-length sequence of period $P_1 = 2^n - 1$, $n \geq 1$. The multiplexed sequence b is defined by

$$b(t) = a(t + \gamma(X(t))), \quad t \geq 0 \tag{5}$$

where $X = (X(t))_{t=0}^{\infty}$ is a periodic integer sequence with period $P_2 = 2^m - 1$, $m \geq 1$, such that $\{0, \ldots, K-1\}$ is the range of values of X, and γ is an injective mapping $\{0, \ldots, K-1\} \rightarrow \{0, \ldots, n-1\}$, $K \geq 2$. Clearly, the multiplexed sequence b can be generated by a multiplexer scheme with X defining the addresses and $\gamma(X)$ defining the stages of the shift register LFSR1 producing a that are used as the inputs to the multiplexer, where X is formed from k stages of another linear feedback shift register LFSR2 with a primitive feedback polynomial, as is suggested in [12]. In this case, $K = 2^k$ for $k \leq m-1$, and $K = 2^k - 1$ for $k = m$.

Multiplexed sequences possess a crosscorrelation weakness: there is a termwise statistical dependence between the output sequence b and the corresponding phase shifts of the sequence a [11]. More precisely, the correlation coefficient between the random variables $b(t)$ and $a(t+\Delta t)$ is equal to the probability that $b(t)$ is chosen to be the same random variable as $a(t+\Delta t)$ for any $\Delta t \in \Gamma$, where $\Gamma = \{\gamma_i : 0 \leq i \leq K-1\}$ and $0 \leq \gamma_0 < \gamma_1 < \cdots < \gamma_{K-1} \leq n-1$. Therefore, for any $\Delta t \in \Gamma$ the correlation coefficient is equal to $1/K$, independently of t. Although K can be made as large as n, this correlation is large enough to be taken into account with respect to the fast correlation attacks.

6.2 Experimental results

In the experiments, we used a 63-bit LFSR with a primitive feedback polynomial, $1+x+x^{63}$, of weight three as LFSR1 and a 59-bit LFSR with a primitive feedback polynomial $1+x^2+x^4+x^7+x^{59}$ of weight five as LFSR2. A set of parity-checks, formed from phase shifts of the parity-check polynomials obtained by the repeated squaring of the LFSR1 feedback polynomial, were used. The experiment was conducted for $K = 4$ and 8. For each K, the experiments were performed for two tapping sequences Γ: consecutive (CONS) and full positive difference set (FPDS). Note that the FPDS tapping sequences are much more resistant to the collision test [2]. For each value of K and tapping sequence, twenty and five different nonzero LFSR initial states were randomly chosen for LFSR1 and LFSR2, respectively.

Our main observation is that the iterative error-correction algorithm, if successful, mainly converges to one of the phase shifts of the sequence a with maximum empirical correlation coefficient.

Table 5 shows the success rate of the performed attacks for various values of the theoretical probability of noise, $p = (1-1/K)/2$, under each set of conditions. As expected, Table 5 shows that the proportion of successful attacks generally decreases as the probability of noise, p, increases; that the success of the attack is practically independent of the tap setting; and that for a given p, increasing the number of parity-checks and the output length increases the success rate. For each tap setting, CONS and FPDS, the success rate of the experiments as a function of the number of taps, K, on LFSR1, the number of polynomial multiples used, $\#PM$, the probability of noise, p, and the keystream sequence length, N, is shown in Table 5.

Table 5: Success rate of fast correlation attack to the multiplexed sequence generator.

K	p	$\#PM$	N	CONS	FPDS
4	0.375	8	5000	52%	68%
4	0.375	8	10000	98%	100%
8	0.4375	10	35000	68%	68%
8	0.4375	11	65000	94%	99%

7 Conclusions

A systematic procedure for applying the fast correlation attack to combiners with memory is presented. This procedure has four main stages. Firstly, linear input and output functions with the maximum possible or comparatively large correlation coefficients are identified, by the LSCA method using the asymptotic probability distribution of the internal state vector (as in the case of summation generator), by searching via the Walsh transform technique (as in the case of nonlinear filter generator), or by applying analytical methods as in the case of the multiplexed sequence generator. In the second stage, the required low-weight parity-check polynomials are formed; in the case of the summation generator, these are based on the input linear function(s) identified in the first stage. In the third stage, the fast correlation attack is applied to the linear transform of the observed keystream sequence and finally, if the attack is successful, the initial state of the input LFSRs are determined, by searching through solutions of linear equations corresponding to the input linear functions determined.

This procedure is successfully applied to several keystream generators: both three and five input summation generators, nonlinear filter generators and multiplexed sequence generators. In each case, the derived maximum correlation coefficients are large enough to apply the fast correlation attack to the observed segment of the keystream sequence, to reconstruct the initial state of the input LFSRs. For success, sufficiently many low-weight parity-check polynomials, corresponding to the input linear function(s) identified in the first stage, have to be generated. In most cases considered, there exist multiple input linear functions with the maximum correlation coefficient correlated to the same output linear function. However, the existence of multiple linear approximations did not confuse the fast correlation attack and the modified keystream sequence converged to one of the best linear functions. Successful experimental results are systematically obtained in all the cases.

References

[1] Specification of the systems of the mac/packet family. technical document 3258-e, October 1986.

[2] R. J. Anderson. Solving a class of stream ciphers. *Cryptologia*, 14(3):285–288, 1990.

[3] R. J. Anderson. Searching for optimum correlation attack. In *Fast Software Encryption - Leuven '94*, volume 1008 of *Lecture Notes in Computer Science*, pages 137–143, 1995.

[4] V. Chepyzhov and B. Smeets. On a fast correlation attack on stream ciphers. In *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science, pages 176–185. Springer-Verlag, 1991.

[5] E. Dawson and A. Clark. Divide and conquer attacks on certain classes of stream ciphers. *Cryptologia*, 18(1):25–40, 1994.

[6] J. Dj. Golić. Correlation via linear sequential circuit approximation of combiners with memory. In R. A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 113–123. Springer-Verlag, 1993.

[7] J. Dj. Golić. Computation of low-weight parity-check polynomials. *Electronics Letters*, 32(21):1981–1982, 1996.

[8] J. Dj. Golić. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, 1996.

[9] J. Dj. Golić. On the security of nonlinear filter generators. In D. Gollmann, editor, *Fast Software Encryption - Cambridge '96*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188, 1996.

[10] J. Dj. Golić, M. Salmasizadeh, A. Clark, A. Khodkar, and E. Dawson. Discrete optimisation and fast correlation attacks. In E. Dawson and J. Golić, editors, *Cryptography: Policy and Algorithms*, volume 1029 of *Lecture Notes in Computer Science*, pages 186–200. Springer-Verlag, 1996.

[11] J. Dj. Golić, M. Salmasizadeh, and E. Dawson. Autocorrelation weakness of multiplexed sequences. In *International Symposium on Information Theory and its Applications 1994*, volume 2, pages 983–987. The Institution of Engineers, Australia, 1994.

[12] S. M. Jennings. *A special class of binary sequences.* PhD thesis, University of London, 1980.

[13] S. M. Jennings. Multiplexed sequences: some properties of the minimum polynomial. In T. Beth, editor, *Proc. Workshop on cryptography, Burg Feuerstein, 1982*, volume 149 of *Lecture Notes in Computer Science*, pages 189–2061. Springer-Verlag, 1983.

[14] S. M. Jennings. Autocorrelation function of the multiplexed sequences. *IEE Proc. F*, 131:169–172, April 1984.

[15] A. Klapper and M. Goresky. Cryptanalysis based on 2-adic rational approximation. In *Advances in Cryptology - CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 262–273. Springer-Verlag, 1995.

[16] J. L. Massey and R. A. Rueppel. Method of, and apparatus for, transforming a digital sequence into an encoded form, u. s. patent no. 4,797,922, 1989.

[17] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–167, 1989.

[18] M. J. Mihaljević and J. Dj. Golić. A comparison of cryptanalytic principles based on iterative error-correction. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 527–531. Springer-Verlag, 1991.

[19] M. J. Mihaljević and J. Dj. Golić. Convergence of a Bayesian iterative error-correction procedure on a noisy shift register sequence. In R. A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 124–137. Springer-Verlag, 1993.

[20] R. Rueppel. *Analysis and Design of Stream Ciphers.* Springer-Verlag, Berlin, 1986.

[21] R. A. Rueppel. Correlation immunity and the summation generator. In *Advances in Cryptology - CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 260–272, 1986.

[22] T. Siegenthaler. Correlation immunity of nonlinear combining functions for cryptographic applications. *IEEE Inform. Theory*, IT-30:776–780, September 1984.

[23] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Comput.*, C-34:81–85, January 1985.

[24] O. Staffelbach and W. Meier. Cryptographic significance of the carry for ciphers based on integer addition. In *Advances in Cryptology - CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 601–614, 1991.

[25] G. Z. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. Inform. Theory*, IT-34:569–571, May 1988.